Webanwendungen mit PHP 4.0 entwickeln

Tobias Ratschiller
Till Gerken

Webanwendungen mit PHP 4.0 entwickeln

An imprint of Pearson Education

München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

Bitte beachten Sie: Der originalen Printversion liegt eine CD-ROM bei.
In der vorliegenden elektronischen Version ist die Lieferung einer CD-ROM nicht enthalten.
Alle Hinweise und alle Verweise auf die CD-ROM sind ungültig.

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Ein Titeldatensatz für diese Publikation ist bei Der Deutschen Bibliothek erhältlich.

Umwelthinweis:
Dieses Buch wurde auf chlorfrei gebleichtem Papier gedruckt.
Die Einschrumpffolie – zum Schutz vor Verschmutzung – ist aus
umweltfreundlichem und recyclingfähigem PE-Material.

Dies ist eine authorisierte Übersetzung der amerikanischen Originalausgabe:
Web Application Development with PHP 4.0, ISBN 0-7357-0997-1, © New Riders Publishing

10 9 8 7 6 5 4 3 2 1

04 03 02 01

ISBN 3-8273-1730-4

Übersetzung: Rebecca Stiels, Diplom-Übersetzerin & Diplom-Technikredakteurin
Einbandgestaltung: atelier für gestaltung, niesner & huber, Wuppertal
Lektorat: Sylvia Hasselbach, shasselbach@pearson.de
Herstellung: Anna Plenk, aplenk@pearson.de
Satz: reemers publishing services gmbh, Krefeld
Druck und Verarbeitung: Bercker Graphische Betriebe, Kevelaer
Printed in Germany

Inhaltsverzeichnis

Teil 2: Webanwendungen

Über die Autoren

Tobias Ratschiller ist Berater für neue Medien in einer italienischen Firma. Er verfügt über umfassende Kenntnisse im Bereich Software-Entwicklung, Datenbank-Design sowie Content Management-Systeme und ist spezialisiert auf die Erstellung von umfangreichen dynamischen Websites. Er installierte einige der weltweit größten Websites oder trug beratend dazu bei. Des weiteren wirkte er bei mehreren Büchern und Artikeln über PHP mit, leitet Seminare in ganz Europa und ist häufig Referent auf führenden Konferenzen.

Till Gerken ist freier Entwickler und Berater für verschiedene Unternehmen, die sich auf die Erstellung von Webanwendungen für Internet-basierte Dienstleistungen spezialisiert haben. Er beherrscht sowohl C/C++, Pascal und x86-Assembler, mit denen er leistungsfähige Multimedia-Systeme, wie 3D-Prozessoren und Echtzeit-Tonmischer, generieren kann, als auch PHP und verwandte Werkzeuge zur Erstellung von mittleren bis großen dynamischen Websites.

Über den Technischen Berater

Mit seinem unschätzbaren Praxiswissen begleitete Graeme Merrall den gesamten Erstellungsprozess des Buches »Web Application Development with PHP 4.0«. Nachdem das Buch geschrieben war, überprüfte er den gesamten Inhalt auf fachliche Korrektheit, Aufbau und Textfluss. Er lieferte wichtiges Feedback, um sicherzustellen, dass »Web Application Development with PHP 4.0« den Ansprüchen der Leser nach aktuellen technischen Informationen genügt.

1993 schloss Graeme sein Biochemie-Studium ab. Bereits während der Studienzeit entdeckte er das Internet, das damals noch in den Kinderschuhen steckte. Dadurch kam er von der Biochemie zu einem Internet-Dienstanbieter und später in ein führendes Unternehmen für Web-Design in Neuseeland. Dort entwickelte er auch seine Fertigkeiten in PHP und ASP.

Neben der Programmierung schrieb Graeme für eine regionale Zeitung in seiner früheren Heimatstadt in Neuseeland. Des weiteren verfasste er mehrere Tutorials und Artikel über PHP für den Web Monkey von Wired Digital.

Geboren und aufgewachsen in Neuseeland lebt Graeme heute in Sydney, wo er seinen eigenen Beratungsdienst leitet, der sich auf E-Commerce und Firmenintegration mittels Internet spezialisiert hat. In seiner Freizeit widmet er sich gerne der modernen Literatur, Musik und dem »Crocodile Wrestling«.

Über Zend Technologies, LTD.

Zend Engine ist der Basis-Skripterstellungsprozessor, der PHP steuert. Der Prozessor ist Eigentum von Zend Technologies, LTD und mit der Q-Public-License für PHP zugelassen. Mit dem Zend-Prozessor wird PHP zu einer leistungsfähigen, zuverlässigen und benutzerfreundlichen Skriptoberfläche.

Die Geschichte von Zend Engine begann vor vier Jahren, als die Firmengründer Zeev Suraski und Andi Gutmans dem Kern-Entwicklerteam von PHP beitraten. Sie entwickelten den Skripterstellungsprozesser von PHP, der inzwischen auf über einer Million Leitrechnern installiert ist. Nach der Einführung von PHP 4.0 hat sich Zend Engine zu einem vielseitigen Skriptprozessor gemausert. Andi Gutmans und Zeev Suraski entwickeln gerade eine Reihe von Produkten, um die Leistungsfähigkeit und den kommerziellen Nutzen von PHP zu erhöhen.

Danksagungen

Ein großes Dankeschön gilt den Mitarbeitern von New Riders. Sie hatten es wahrscheinlich schwer mit uns, und wir möchten uns wirklich für Ihre Freundlichkeit und Professionalität bedanken. Auch an Robin Drake, unserer Herausgeberin besonderen Dank für ihre Geduld mit uns. Außerdem bedanken wir uns bei unserem technischen Berater Graeme Merrall und unserer Herstellerin Ann Quinn.

Die folgenden Leute haben uns in den verschiedenen Phasen der Buchererstellung geholfen. Auch bei ihnen möchten wir uns bedanken: Alan Bower, Nancy Maragioglio, Jakob Nielsen, Kristian Koehntopp, Zeev Suraski, Andi Gutmans, Leon Atkinson, Alexander Aulbach, Uwe Steinmann, Boaz Yahav und Rafi Ton. Ebenso bei den Verfassern unserer Fallstudien. Zuletzt bedanken wir uns bei SoftQuad für die Bereitstellung ihres hervorragenden XMetaL-XML-Editors, mit dem wir den Text erstellen und bearbeiten konnten.

Danksagungen von Tobias

Die Person, welcher der größte Dank gebührt, ist natürlich Till Gerken, der ein großartiger Co-Autor war. Wir haben zusammen Tausende von Stunden (oder so ähnlich) mit Internet Relay Chat (IRC) zugebracht, in denen wir Kapitel überarbeiteten, Programmcode schrieben oder korrigierten – und viel Spaß hatten. Es war richtig harte Arbeit, aber auch eine wunderbare Zeit.

Danke an alle Benutzer des Efnet's #php – sie sind eine großartige Gemeinde, und es macht Spaß, sich unter ihnen zu tummeln. Wenn Ihr Zeit habt, kommt vorbei und sagt »Hallo« zu tigloo (Till) und Yapa (das bin ich). Jeder im IRC war sehr hilfsbereit, und insbesondere Zeev zeigte sehr viel Geduld bei der Beantwortung unserer Fragen.

Ein Dankeschön auch an Robert Finazzer, der uns in den letzten Jahren wertvoller Wirtschaftsberater war und stets Verständnis hatte, wenn ich Artikel oder Bücher schrieb, anstatt mit ihm an millionenschweren Unternehmungen zu arbeiten. Grüße an das übrige Team des Profi Online Service und natürlich an Joachim Marangoni.

Danksagungen von Till

Ich habe schon nicht mehr daran geglaubt, dass ich jemals bis zu diesem Kapitel vordringen würde, aber nun ist es soweit. Damit beende ich ein Projekt, für das ich im letzten Jahr viel Zeit und Energie aufgebracht habe. Ich muss zugeben, dass ich manchmal dachte, das schaffen wir nicht; aber jetzt bin ich stolz, es im Regal stehen zu sehen.

Deswegen muss ich mich als Erstes bei Tobias Ratschiller bedanken, der mich in den PHP-Bereich einführte. Von Anfang an hatte er ein unglaubliches Vertrauen in mich und bewies bei der Zusammenarbeit mit mir unendliche Geduld. Er war ein fünf-Sterne-Co-Autor, und ich bin froh, dass ich die Möglichkeit hatte, dieses Buch mit ihm zusammen zu schreiben. Selbst wenn ich manchmal mit meinem Material unzufrieden war, wartete er immer mit guten Vorschlägen auf. Wie er bereits sagte, verbrachten wir endlose Stunden im IRC mit der gegenseitigen Textkontrolle und Fehlerkorrektur des Programmcodes – ganz zu schweigen von den unzähligen Emails. Es hat definitiv viel Spaß gemacht!

Zusätzlich zu den oben erwähnten Danksagungen möchte ich mich bei meinen Freunden bedanken: bei denen, die mich in meiner Arbeit unterstützten, und bei denen, die das nicht taten. Ich habe es zwar immer gehasst, wenn andere Autoren bemerkten, es seien zu viele, um sie alle namentlich zu erwähnen, aber ich muss zugegeben, dass es mir genauso geht und ich mehr Schaden anrichten würde, wenn ich jemanden ausließe. Ihr wisst, wer gemeint ist!

Zu guter Letzt möchte ich meiner Familie für Ihre Aufmunterungen und tatkräftige Unterstützung bei meiner Arbeit danken. Sie gaben mir stets ein Zuhause, wenn ich einen Platz zum Ausruhen brauchte.

Sagen Sie uns, was Sie denken

Als Leser dieses Buches sind Sie unser wichtigster Kritiker und Kommentator. Wir schätzen Ihre Meinung und möchten gerne wissen, was richtig war, was wir besser machen können und an welchen anderen Bereichen Sie interessiert sind. Wenn Sie möchten, geben Sie uns auch andere weise Ratschläge.

Als leitender Herausgeber bei New Riders Publishing begrüße ich Ihre Kommentare. Sie können mir faxen, eine Email schicken oder direkt schreiben, um mir mitzuteilen, was Ihnen an diesem Buch gefallen hat und was nicht – und um mir zu sagen, was wir tun können, damit unsere Bücher noch besser werden.

Bitte beachten Sie, dass ich Ihnen bei technischen Problemen, die mit dem Thema dieses Buches zusammenhängen, nicht weiterhelfen kann. Aufgrund der Masse der Emails, die ich erhalte, kann ich möglicherweise auch nicht jede Nachricht beantworten.

Wenn Sie mir schreiben, denken Sie daran, den Titel und den Autor dieses Buches sowie Ihren Namen und Ihre Telefonnummer bzw. Fax-Nummer anzugeben. Ich werde Ihre Anmerkungen gründlich überprüfen und sie mit dem Autor und Herausgebern erörtern.

Fax: 317-581-4663

Email: nrfeedback@newriders.com

Post: Al Valvano
Executive Editor
New Riders Publishing
201 West 103rd Street
Indianapolis, IN 46290 USA

Vorwort von Zeev Suraski

Als ich vor drei Jahren das erste Mal mit PHP konfrontiert wurde, hatte ich keine Ahnung, dass ich eines Tages ein Vorwort für ein PHP-Buch schreiben würde. Tatsächlich schien mir damals die Vorstellung, dass es jemals ein Buch über PHP geben würde, ein wenig weit hergeholt. Rückblickend ist das, was PHP zu einem der am weitesten verbreiteten Skriptsprachen für die Web-Entwicklung machte, nur erstaunlich. Meine Mitwirkung am PHP-Projekt begann, wie viele Dinge, zufällig. Ich stolperte als Endbenutzer über einen Fehler in PHP/FI 2.0 – etwas, das seltsam genug war, damit mein Kollege Andi Gutmans und ich uns die Sache näher betrachteten. Als wir uns in den Programmcode vertieften, der PHP/FI 2.0 zum Laufen brachte, waren wir nicht sehr begeistert. Auf der anderen Seite gefiel uns die Vorstellung einer in HTML eingebetteten, servergestützten Skriptsprache für die Serverseite. Als gute zukünftige Software-Entwickler beschlossen wir, den Code noch einmal von Grund auf neu zu erschaffen, aber diesmal richtig.

Durch das Umschreiben der Programmiersprache und die Unmenge an »Zuarbeiten«, die in Form von zahlreichen Funktionsmodulen und Beispielcode geleistet wurden, erhielt PHP einen Auftrieb, der unsere wildesten Träume und Erwartungen sprengte. Heute wird PHP in mehr als einer Million Domänen im Internet benutzt und ist das Werkzeug der Wahl für die Skripterstellung in UNIX-Umgebungen auf der Server-Seite. PHP 4.0 stellt die logische Fortsetzung dar, die sicherstellt, dass PHP auf Jahre hinaus die führende Technologie im Bereich der Web-Skripte bleibt. Der Zend-Prozessor (www.zend.com) revolutioniert die Leistungsfähigkeit und Skalierbarkeit von PHP-basierten Websites. Durch die integrierte Unterstützung für Sitzungen, XML, Java und COM sowie eine Reihe weiterer Funktionen können Web-Entwickler leichter als jemals zuvor noch leistungsfähigere, dynamische Websites entwickeln.

Durch die kontinuierliche Entwicklung und Integration führender Technologien bleibt PHP stets aktuell. Die neue Java- und DCOM-Unterstützung, die erweiterten XML-Funktionen und die verbesserten OOP-Funktionen erhöhen die Akzeptanz von PHP in der Geschäftswelt und machen es zu einem wertvollen Werkzeug für die Datenverarbeitung im Unternehmen. Die kommerziellen AddOns von Zend Technologies – zum Beispiel Debugger, DIE und Compiler – brachte die Firma einen weiteren Schritt voran. Auch das Innenleben der PHP 4.0-Architektur wurde revolutioniert, was jedoch die meisten Endbenutzer nicht bemerken werden. Die Webserver-Schnittstelle wurde beispielsweise vollständig abstrahiert, so dass auch andere Web-Server als Apache unterstützt werden können. Bücher wie dieses versorgen Sie mit den notwendigen Hintergrundinformationen, damit Sie diese neuen Technologien erfolgreich einsetzen können.

Nach meiner Ansicht sieht die Zukunft für Open Source im Allgemeinen und PHP im Besonderen sehr gut aus. 1997 mussten Sie durch Reifen springen, um Ihren Geschäftsführer davon zu überzeugen, dass Linux mindestens so stabil ist wie Windows NT, und die Verwendung von Open Source in großen Unternehmen wurde noch nicht einmal in Betracht gezogen. Die Welt hat sich geändert. Firmen, die es sich zur Aufgabe gemacht haben, Linux-basierte Lösungen, wie etwa RedHat, SuSE und VA Linux zu unterstützen, wurden nicht nur Wirtschaftsgiganten, sondern schafften es auch, Linux und Open Source als akzeptable Lösung in Firmen einzuführen. Zum Glück waren diese Firmen klug genug, dabei den Geist von Open Source und den Kontakt zu der Benutzer-Gemeinde aufrecht zu erhalten. Das Open Source-Entwicklungskonzept auf der einen Seite und die starke kommerzielle Vermarktung auf der anderen verschafften Linux einen unvorhergesehenen Aufschwung. Ich bin sicher, dass kommerzielle Betriebe, wie Zend Technologies, die sich der Unterstützung von PHP verschrieben haben, dazu beitragen, dass die Verwendung von PHP noch weitere Verbreitung findet, insbesondere auf High-End-Websites.

Ich möchte die Gelegenheit ergreifen, Herrn Prof. Michael Rodeh von IBM Haifa und des Technion Institute of Technology zu danken, der Andi und mich ermunterte, Rasmus Lerdorf zu kontaktieren, den Autor von PHP / FI 1.0 und 2.0. Rasmus Lerdorf freute sich sehr, mit uns zusammen PHP 3.0 zu entwickeln, den offiziellen Nachfolger von PHP / FI 2.0. Des weiteren bedanke ich mich bei der PHP-Gruppe, dem gesamten PHP-Entwicklerteam, ohne das PHP nicht das hervorragende Werkzeug wäre, das es heute ist, und schließlich der PHP-Gemeinde, die sich als unerschöpfliche Quelle für Ideen und Unterstützung erwies.

Ich bin sicher, dass Ihnen dieses Buch hilft, etwas über das erweiterte PHP und die Entwicklung von Webanwendungen zu erfahren. Es ist eins der wenigen Bücher, die über die reine Beschreibung der Syntax hinausgeht. Es bringt Ihnen die Konzepte hinter der Programmiersprache nahe und kann Ihnen helfen, Ihre Problemlösungsstrategien für die Web-Programmierung zu verbessern.

Viel Glück!

Zeev Suraski

Einführung

In der Antike versuchten die alten Meister nicht, die Menschen zu erziehen, sondern brachten ihnen bei, nicht zu wissen.

Der Erfolg der Open Source-Software, wie Linux oder Apache, wurde in allen großen Medien ausführlich dokumentiert. Trotzdem blieb der Aufstieg von PHP größtenteils unbemerkt. Gemäß einer E-Soft-Umfrage (`www.e-softinc.com/survey/`) ist die Web-Skriptsprache PHP immer noch das verbreitetste Modul für den Apache-Webserver. Netcraft-Studien haben ergeben, dass PHP auf über 6% aller Web-Domänen in der Welt (siehe `www.netcraft.com/survey`) verwendet wird. Dies ist eine unglaubliche Marktdurchdringung für ein so spezialisiertes Produkt. Und die Beliebtheit steigt weiter an. Dies zeigt sich verstärkt in den traditionellen Medien. Bis zum Mai 2000 wurden über 20 Bücher über PHP in verschiedenen Sprachen veröffentlicht, und weitere sind in Vorbereitung.

Immer mehr kommerzielle Anbieter springen auf den Zug auf: PHP wird auf Webservern integriert, zum Beispiel C2's Stronghold und Linux-Distributionen. Eine neue Firma namens Zend Technologies wurde gegründet, die kommerzielle AddOns und Unterstützung für PHP entwickelt. Eine lange Liste von großen Websites benutzen PHP ebenso wie Hunderttausende kleiner bis mittlerer Websites.

Für uns Autoren begann dieses Buch im Juni 1999, als New Riders Publishing mit der Bitte an uns herantrat, ein Buch über das erweiterte PHP zu schreiben. Die Idee, ein PHP-Buch zu schreiben, geisterte bereits seit einiger Zeit in unseren Köpfen herum, und das Angebot von New Riders kam sehr willkommen.

Nach über 1.500 Emails, 500 überarbeitete Dateien und unzähligen Stunden im IRC waren wir schließlich fertig. Es war Schwerstarbeit, aber ich glaube, das Ergebnis ist mehr als nur ein Referenzhandbuch. Wir haben uns nicht darauf beschränkt, einen bloßen Überblick über die PHP-Funktionen zu geben, sondern versucht, die Konzepte zur Entwicklung von Webanwendungen zu erläutern.

Die Entwicklung vom unerfahrenen Programmierer, der keine oder nur eine geringe fachliche Ausbildung hat, bis hin zum Software-Entwicklungsexperten geschieht in mehreren Schritten. Der Programmierer beginnt seine Karriere als Lehrling. Zu diesem Zeitpunkt interessiert er sich in der Regel nicht für Programmierstile, Planung oder Testphasen – unleserlicher Code, fehlende Sicherheit und lange Hacker-Nächte sind in dieser Phase typisch. Möglicherweise kennt der Programmierer alle Tricks und versteckten Funktionen einer Programmiersprache, aber er hat Probleme bei der Entwicklung und Daten-

pflege im Team und mit größeren Projekten. An diesem Punkt können Sie leicht erkennen, wer wirklich das Zeug zu einem guten Programmierer hat. Dieser stellt sich folgende Fragen:

- Wie kann ich es umgehen, dieselben Funktionen immer von neuem eingeben zu müssen?
- Welche Vorkehrungen muss ich treffen, um meine Anwendung sicher und stabil zu machen?
- Was muss ich tun, damit meine Anwendung leichter zu pflegen ist?
- Wie können mehrere Leute effizient in einem Team zusammenarbeiten?

An dieser Stelle kommt unser Buch ins Spiel. Wir hoffen, Software-Entwicklern Richtlinien für eine bessere Entwicklung von PHP und Webanwendungen geben zu können. Heute sind viele Technologien verfügbar. Diese können Sie nur vollständig nutzen, wenn Sie die grundlegenden Prinzipien verstehen, die hinter dem Entwicklungsprozess stehen und wenn Sie Problemlösungsstrategien entwickeln. Typische Referenzhandbücher helfen bei diesen Fragen nicht weiter.

Zielgruppe

Wenn Sie Programmier-Laie sind, ist dieses Buch nicht für Sie geeignet. Nutzen werden Sie wahrscheinlich in einem der folgenden Fälle aus diesem Buch ziehen können:

- Sie haben bereits Anwendungen mit PHP entwickelt und möchten Ihre Fertigkeiten ausbauen.
- Sie haben Erfahrung mit anderen Programmiersprachen und möchten Webanwendungen mit PHP entwickeln.
- Sie sind PHP-Experte und möchten Ihre PHP-Kenntnisse erweitern.

Sie müssen kein PHP-Guru sein, um dieses Buch zu lesen, aber Sie sollten mit der Syntax von PHP vertraut sein oder sich mit Programmierprinzipien auskennen.

Voraussetzungen

Dieses Buch geht davon aus, dass Sie eine funktionierende PHP-Installation haben, vorzugsweise PHP 4.0 oder höher. Aufgrund seiner Beliebtheit verwenden wir bei Bedarf als Datenbanksystem MySQL. Da Plattformunabhängigkeit eine der stärksten Vorzüge von PHP ist, sollten unsere Beispiele sowohl unter UNIX als auch unter Windows laufen.

Aufbau dieses Buches

Dieses Buch ist in drei Teile unterteilt. Teil I »Erweitertes PHP« befasst sich mit der erweiterten Syntax von PHP, z.B. der Objektausrichtung, den dynamischen Funktionen und Variablen und dem selbstmodifizierenden Programmcode. Er gibt einen Überblick über Projektplanungsprinzipien, Programmierstile und Programmdesign. Dieser Teil bietet Ihnen die notwendige Grundlage für eine schnelle und produktive Entwicklung von industriellen Webanwendungen.

Teil II »Webanwendungen« konzentriert sich auf die Entwicklung der Software. Er geht darauf ein, warum Sitzungen wichtig sind, welche Sicherheits-Richtlinien Sie beachten sollten, inwiefern Tauglichkeit von Bedeutung ist und wie Sie PHPLib für die Sitzungsverwaltung und den Datenbankzugriff nutzen können. Hier finden Sie auch drei Fallstudien von aufeinanderfolgenden PHP-Projekten, mit denen Sie Ihren IT-Manager überzeugen können.

Teil III des Buches »Über PHP hinaus« ist für jene Leser, die mehr wissen wollten, als was derzeit mit PHP möglich ist. Er zeigt, wie PHP mit C erweitert werden kann. Dies ist die offizielle Dokumentation über die Erweiterung von PHP, die durch Zend Technologies anerkannt wurde.

Im Detail werden folgende Themen behandelt:

Kapitel 1 – Entwicklungskonzepte

Wenn Sie anspruchsvolle Projekte abwickeln müssen, ist die Verwendung von Kodierungskonventionen, richtige Planung und eine erweiterte Syntax ein absolutes Muss. Dieses Kapitel behandelt die allgemeinen Kodierungskonventionen, die für alle industriellen Projekte unerlässlich sind (Namens- und Kommentarkonventionen), und erläutert, wie der Quellcode in logische Module unterteilt werden kann.

Kapitel 2 – Erweiterte Syntax

Dieses Kapitel befasst sich mit der erweiterten PHP-Syntax, z.B. mehrdimensionale Arrays, Klassen, variable Variablen, selbstmodifizierender Programmcode und ähnliches.

Kapitel 3 – Programmdesign: Ein Praxisbeispiel

In diesem Kapitel gehen wir den gesamten Prozess zur Planung einer kompletten Webanwendung durch: mit phpChat, einer Web-basierten Chat-Client-Schnittstelle für IRC. Dieses Kapitel führt die Planungsgrundlagen an, gibt Richtlinien zur Projektorganisation und zeigt, wie modulare pluginfähige Anwendungen entwickelt werden.

Kapitel 4 – Webanwendungskonzepte

Sitzungsverwaltung, Sicherheitsaspekte und Genehmigung sowie Tauglichkeit bilden die Grundlage für alle Webanwendungen. Ohne die richtige Sitzungsverwaltung lassen sich Webanwendungen nicht ausführen. Sie müssen einen Weg finden, um Benutzer während einer mehrseitigen Anfrage zu erkennen, wenn Sie Variablen, wie einen Einkaufskorb, einem speziellen Benutzer zuordnen möchten. Diese Identifizierung sollte so sicher sein wie möglich, wenn Sie nicht wollen, dass ein Benutzer die Kreditkarteninformationen eines anderen Benutzers sieht. Tatsächlich sind spezielle Erwägungen erforderlich, um die Sicherheit in Ihren Anwendungen zu erhöhen. Auch wenn PHP für Hackerattacken weniger anfällig ist als andere CGI-Umgebungen, kann es sehr leicht passieren, dass Sie vollkommen ungeschützte Anwendungen schreiben, wenn Sie einige wichtige Prinzipien außer Acht lassen, die in diesem Kapitel behandelt werden.

Dieses Kapitel führt außerdem die grundlegenden Tauglichkeitskonzepte ein. Sobald wir nicht mehr von unabhängigen Skripten, sondern von Anwendungen reden, fällt die Rolle des Benutzers mehr ins Gewicht. Immerhin ist es der Benutzer, der über den Erfolg oder Misserfolg eines Projekts entscheidet. Dieses Kapitel zeigt Ihnen, wie Sie eine höhere Zufriedenheit beim Benutzer erreichen.

Kapitel 5 – Grundlegende Webanwendungsstrategien

Dieses Kapitel erörtert weitere Grundlagen für Webanwendungen. Alle Webanwendungen verarbeiten beispielsweise Formulareingaben oder trennen zwischen Layout und Programmcode. Weiterhin führt Sie dieses Kapitel in die effektive Teamarbeit ein, indem es einen Überblick über Versionskontrolle mit CVS gibt. Schließlich werden mehrstufige Anwendungen, COM und Java aus PHP-Sicht besprochen.

Kapitel 6 – Datenbankzugriff mit PHP

Ohne Datenbanken gibt es keine Webanwendungen. Kapitel 6 stellt PHPLib als Werkzeug für einen herstellerunabhängigen Datenbankzugriff vor und gibt einen Überblick über seine anderen Funktionen, wie Sitzungsverwaltung, Benutzerberechtigung und Genehmigungsverwaltung.

Kapitel 7 – Anwendungen der Spitzentechnologie

Durch Entwicklung eines vollständigen Wissensbestands mit PHPLib macht Sie dieses Kapitel mit PHPLib's Schablonenklasse, Selbstbezüge in SQL und anderen anspruchsvollen Themen vertraut. Anschließend gibt dieses Kapitel einen Überblick über XML und zeigt, wie Anwendungen von dieser aufregenden Technologie profitieren können. Außerdem beschreibt dieses Kapitel die Schnittstelle von PHP für die XML-Syntaxanalyse und seine WDDX-Funktionen.

Kapitel 8 – Fallstudien

Erfolgsberichte können extrem hilfreich sein, wenn eine neue Technologie in einer Firmenumgebung eingeführt wird. In Kapitel 8 stellen wir Fallstudien vor, in denen Six Open Systems, BizChek und Marketplayer.com vorkommen – drei große Beispiele aus Hunderten von Firmen, in denen PHP erfolgreich in anspruchsvollen Szenarios verwendet wurde.

Kapitel 9 – Erweiterung von PHP 4.0: Erweiterung des Kerns von PHP

Reichen mehr als 1.200 Funktionen für Sie immer noch nicht aus? Kein Problem, denn dieses Kapitel ist die offizielle Dokumentation für die Erweiterung von PHP. Wenn Sie sich etwas mit C auskennen, gibt Ihnen Kapitel 9 einen komprimierten Einblick in die Interna von PHP 4.0 und zeigt Ihnen, wie Sie Ihre eigenen Module schreiben, um PHPs Funktionalität zu erweitern.

In diesem Buch verwendete Konventionen

Folgende Konventionen werden in diesem Buch verwendet:

Konvention	Verwendung
Kursiv	Neue Begriffe werden definiert.
`Dicktengleiche Schrift`	Befehle, Syntaxzeilen usw. sowie Internetadressen, wie z.B. `www.phpwizard.net`.
➥	Die Syntax läuft über mehr als eine Zeile, die so markierte Zeile setzt die vorangegangene fort.

Teil 1:
PHP für Fortgeschrittene

1. Entwicklungskonzepte
2. Erweiterte Syntax
3. Programmentwicklung: ein Praxisbeispiel

Pearson
Education

ADDISON-WESLEY

1 Entwicklungskonzepte

Ohne Namen bist Du ein Niemand.

Um eine Sprache wirklich zu beherrschen, müssen Sie nicht nur die Syntax und die Semantik der Sprache verstehen, sondern auch ihre Philosophie, ihren Hintergrund und die Entwurfsprinzipien kennen.

1.1 Ist PHP etwas für mich?

Haben Sie sich jemals gefragt, warum es so viele Programmiersprachen gibt? Neben den »Hauptsprachen«, wie C, C++, Pascal und ähnlichen gibt es andere, wie Logo, Cobol, Fortran, Simula und viele weitere exotische Sprachen. Viele Software-Entwickler denken beim Anlegen eines Projekts nicht ernsthaft über alternative Programmiersprachen nach. Sie haben ihre bevorzugte Sprache (vielleicht auch eine vom Unternehmen vorgeschriebene Sprache), kennen ihre Vorteile und ihre Nachteile und passen das Projekt an die speziellen Stärken und Schwächen der Sprache an. Damit bürden Sie sich jedoch möglicherweise unnötige zusätzliche Belastungen auf, um die Mängel der gewählten Sprache auszubügeln.

Eine Sprache zu benutzen, ohne ihre spezifischen Konzepte zu kennen, ist so, als wolle man als Lastwagenfahrer an einem Autorennen teilnehmen. Im Prinzip wissen Sie, wie Sie einen Wagen steuern müssen, möglicherweise erreichen Sie auch einen guten Platz, aber ein herausragender Fahrer werden Sie erst, nachdem Sie sich mit den Besonderheiten Ihres neuen Vehikels vertraut gemacht haben.

Auf ähnliche Weise wird der objektorientierter Programmierer, der ein Programm schreiben soll, versuchen, das Programm in Objekte einzupassen, während ein auf Prozeduren spezialisierter Programmierer dieselbe Aufgabe anders realisiert. Welcher Ansatz ist besser? Jeder Programmierer wird sagen, dass seine Methode die beste sei, aber nur jemand, der mit beiden Konzepten vertraut ist – objektorientierte Programmierung und prozedurale Programmierung – kann es wirklich beurteilen.

Jede der erwähnten Sprachen verfolgt einen anderen Ansatz, um Probleme auf spezifische Weise zu lösen – meistens nur Probleme einer speziellen Art, mit spezifischen Anforderungen. Da sich diese Sprachen auf ein sehr begrenztes Anwendungsgebiet konzentrieren, ist auch der Erfolg auf diese Gebiete begrenzt. Sprachen wie C und Pascal errangen wahrscheinlich deshalb soviel

Beliebtheit, weil sie auf einem breitgefächerten Gebiet arbeiten. Spezielle Funktionen für spezifische Probleme gibt es zwar nicht, aber allgemeine Probleme lassen sich lösen.

Wie passt PHP in dieses Schema? Sie wird zwar als Sprache bezeichnet, aber genaugenommen ist es keine eigenständige Sprache, sondern eine Mischung mehrerer Sprachen. Sie verwendet hauptsächlich die Syntax, welche die meisten Programmierer aus C kennen; trotzdem ist sie komplett anders, weil sie interpretiert wird. PHP kennt verschiedene Variablentypen, nimmt aber keine strenge Überprüfung von Typen vor. Des weiteren sind in PHP Klassen, aber keine strukturierten Typen bekannt. Es gibt eine Vielzahl weiterer Beispiele wie dieses, aber das Wesentliche haben Sie wahrscheinlich erfasst: PHP vereint verschiedene konzeptuelle Ansätze in einem einzigartigen, neuen Ansatz.

Um Web-Anwendungen erfolgreich mit PHP zu erstellen, sollten Sie zunächst die folgende Frage beantworten: Ist PHP die ideale Sprache für mein Projekt? Gute Frage. Wir wären dumm, wenn wir »nein« sagten. (Wer schreibt schon ein Buch über etwas, das er für schlecht hält?) Lassen Sie uns die Frage neu formulieren: Gibt es eine bessere Sprache als PHP für mein Projekt? Diese Frage können Sie sicher beantworten. Wenn Sie Web-Anwendungen entwickeln, ist PHP die geeignete Sprache für Sie.

1.2 Die Bedeutung der Planung

Warum Sie diesen Abschnitt lesen sollten

Selbst wenn Sie bereits ein professioneller Programmierer sind, der mit PHP vertraut ist, empfehlen wir Ihnen, die folgenden Abschnitte zu lesen, das sie die Grundlagen für eine erfolgreiche Entwicklung enthalten. Auch wenn Sie sich bereits mit den angesprochenen Themen auskennen, sollten Sie sich die Zeit nehmen, den Text durchzublättern; Sie könnten neue Informationen entdecken – andere Ansichten, andere Ansätze andere Lösungen. Je mehr Sie über die Herangehensweise an die verschiedenen Aspekte Ihrer künftigen Projekte wissen, um so besser sind Sie in der Lage, die kritischen Teile exakt zu bestimmen und sie gekonnt zu verarbeiten. Viele der folgenden Abschnitte erörtern auch Themen, die mehr eine Ansichtsfrage sind als allgemein anerkannte Regeln. Vertrauen Sie uns als professionellen Entwicklern und verlassen Sie sich auf unserer Erfahrung, bevor Sie den Inhalt verwerfen – es wird sich später auszahlen.

Bevor wir in PHP-spezifische Themen eintauchen, lassen Sie uns auf einer breiteren Basis beginnen. Einige Themen beziehen sich auf die Programmierung von Anwendungen im Allgemeinen, ungeachtet der Sprache, die Sie einsetzen oder der Plattform, auf der Sie sich befinden.

Wenn Sie an einem professionellen Projekt arbeiten, ist es sehr wichtig, dass Sie sich überlegen, was Sie tun. Informieren Sie sich über Ihre Feinde – und unterschätzen Sie diese niemals! Natürlich ist ein Projekt kein Feind, aber die Aussage gilt trotzdem. Machen Sie sich mit allen Spezifikationen Ihres Projekts vertraut, der / n Zielplattform(en) und seinen Benutzern, und unterschätzen Sie niemals die Bedeutung der kleinen Problemchen, die Sie nicht vollständig abgeschätzt haben, bevor Sie sich anderen Themen zuwenden.

Nach unserer Erfahrung beansprucht die Planung mindestens 50% der Entwicklungszeit. Je größer ein Projekt ist, umso gründlicher sollten Sie sich darauf vorbereiten. Dieses Prinzip gilt sowohl, wenn Sie Ihren Kunden kontakten und mit ihm zusammen den Rahmen für das Projekt ausarbeiten, als auch, wenn Sie mit Ihren Entwicklern sprechen, um den Programmcode zu umreißen. Je weniger Mühe Sie auf Konsistenz und Pflege verwenden, umso eher werden Sie Probleme bekommen, wenn Sie alte Dateien wieder öffnen müssen, um Fehler zu entfernen oder neue Funktionen hinzuzufügen.

Die Zeit für die Planung muss nicht unbedingt proportional zur Größe des Projekts sein. Denken Sie zum Beispiel an einen Suchalgorithmus, den Sie entwerfen sollen. Das Programm muss nicht mehr tun, als sich durch einen Haufen von Informationen durchzugraben und Daten nach bestimmten Regeln auszusortieren. Angenommen, die Daten seien bereits vorhanden, so dass die Einrichtung und die Ausgabe keine große Mühe kostet. Den größten Teil der Zeit wird das Programm in seiner Haupt-Suchschleife verbringen. Diese Schleife benötigt wahrscheinlich nicht mehr als 100 Zeilen Programmcode, aber das Auswählen bzw. Entwickeln eines optimalen Algorithmus kann leicht einen ganzen Tag dauern. Diese kleine Schleife erfordert möglicherweise die meiste Zeit Ihrer Planung, während Sie auf der anderen Seite Projekte mit einigen Tausend Zeilen in weniger als einem Tag durchgeplant haben.

Ein zweites Beispiel: Angenommen, Sie benötigen ein kleines Skript, das alle Dateien eines Verzeichnisses auflistet. Sie könnten den Programmcode einfach runterhacken, so dass es genau und nur diese spezielle Aufgabe erfüllt und alle Dateien in einem angegebenen Verzeichnis auflistet. Dann müssen Sie sich nicht mehr darum kümmern. Das Problem ist gelöst, und Sie können zu anderen Aufgaben übergehen. Eine andere Strategie könnte sein, dass Sie sich überlegen, ob Sie zu einem späteren Zeitpunkt – vielleicht in einem ganz anderen Projekt – dies oder ein ähnliches Werkzeug noch einmal benötigen. Jedes Mal eine Routine zum Auflisten von Verzeichnisinhalten zu schreiben, die jeweils auf eine ganz spezielle Aufgabe zugeschnitten ist, wäre Zeitverschwendung. Denken Sie darüber nach, wenn Sie in solch eine Situation kommen. Sie könnten beispielsweise ein separates Modul schreiben, mit dem Sie verschiedene Verzeichnisse auflisten und optional Unterverzeichnisse mehrfach aufrufen, und möglicherweise auch Platzhalter verwenden. Sie könnten eine kleine, abgesicherte Funktion definieren, welche die meisten Spezialfälle

abdeckt und gleichzeitig die alltäglichen Anforderungen einer Verzeichnisauflistungsfunktion erfüllt. Mit letzterem Verfahren hätten Sie nach ein paar Projekten eine kleine Bibliothek an robusten Programmierwerkzeugen, die Sie bedenkenlos wieder einsetzen und denen Sie vertrauen können. Dies könnte die Entwicklungszeit in künftigen Projekten erheblich verkürzen.

Die Anzahl der kostenlos erhältlichen Werkzeugbibliotheken wird zwar immer größer, aber das wird kaum ausreichen, um all Ihre Bedürfnisse abzudecken; sie werden auch nicht unbedingt für Ihre Probleme geeignet sein. Einige Bibliotheken sind außerdem zu groß, um sie stets mit sich rumzutragen – wenn Sie für jeden Treffer Hunderte von Kilobytes Programmcode durchforsten müssen, wird die Leistungsfähigkeit Ihres Standorts erheblich reduziert. In diesem Fall zahlt es sich möglicherweise aus, eine sub-optimale Lösung durch eine 100% optimale Lösung zu ersetzen, die Sie selbst entwickelt haben.

Bei größeren Projekten ist die Gefahr von Problemen durch mangelnde Planung noch größer. Im letzten Drittel der Entwicklung stoßen Sie plötzlich auf Schwierigkeiten, die Sie nicht vorhergesehen haben, weil Sie zu wenig Zeit für die Planung aufgewendet haben. Diese Probleme können so schwerwiegend sein, dass Sie das gesamte Projekt neu strukturieren müssen. Stellen Sie sich ein datenbankgestütztes Programm vor, dass sich auf eine zusätzliche Datenbank-Abstraktionsebene stützt. Diese Abstraktionsebene lässt nur Textdaten zu. Später stellen Sie aber fest, dass Sie auch numerische Daten benötigen. Möglicherweise schaffen Sie es, das Problem über einen kleinen Umweg zu lösen, müssen aber feststellen, dass dieser Umweg Ihren Anforderungen nicht genügt. Das Einzige, was Sie an diesem Punkt noch tun können, ist, die Datenbank-Schnittstelle zu ändern. Dazu müssen Sie sowohl die Abstraktionsebene ändern als auch alle Stellen im Hauptprogramm überprüfen, an denen diese Ebene aufgerufen wird – und natürlich den zuvor geschaffenen Umweg entfernen.

Dies sind Arbeitsstunden oder -tage, die Sie von Anfang an hätten vermeiden können – Probleme, die häufig zwischen Erfolg und Misserfolg entscheiden, da Zeit die wertvollste Ressource ist, von der Sie niemals genug haben.

Die folgenden Abschnitte leiten Sie durch die meisten grundlegenden und trotzdem sehr wichtigen praktischen Aspekte der Entwicklung: Verbesserung der Programmcode-Qualität sowie fundamentale Entwurfs- und Dokumentationsaspekte. Anschließend erstellen wir eine Anwendungsprogrammschnittstelle (API), wobei wir einen direkten, praktischen Ansatz verfolgen, um Sie mit den neuen Konzepten vertraut zu machen, bevor wir eine weitere API von Grund auf neu entwickeln – d.h. sie zunächst »theoretisch« auf dem Papier entwerfen und dann ein paar praktische Prinzipien ausarbeiten, die Ihnen bei der Implementierung Ihrer nächsten API helfen – Stilaspekte, unbedingt zu beachtende und zu vermeidende Punkte sowie ein paar Tricks.

1.3 Kodierungskonventionen

Was ist der Unterschied zwischen gutem und schlechtem Programmcode? Eigentlich sehr einfach. Guter Programmcode – wirklich guter Code – lässt sich fast wie ein Buch lesen. Sie können an einer beliebigen Stelle anfangen und wissen sofort, wozu die Zeilen dienen, die Sie lesen, unter welchen Umständen sie ausgeführt werden und welche Einstellungen möglicherweise notwendig sind. Selbst wenn Sie über kein Hintergrundwissen verfügen und auf einen anspruchsvollen und komplizierten Algorithmus stoßen, können Sie zumindest schnell feststellen, welche Aufgabe er hat und wann er ausgeführt wird.

Es wäre einfach, Beispiele aufzuführen und zu sagen »Machen Sie es so«, aber wir möchten, dass Ihnen dieses Kapitel eine solide Grundlage für die Erstellung professionellen Programmcodes gibt. Es soll den Unterschied zwischen gründlich durchdachtem Programmcode und gewöhnlicher Hackerei zeigen. Es würde den Rahmen sprengen, alle Aspekte guten Programmierens so intensiv zu erörtern, wie wir es gerne würden, aber dieses Kapitel vermittelt Ihnen einen guten Start. Wir empfehlen Ihnen dringend, sich speziellere Literatur zu besorgen, um sich mit den Einzelheiten des Software-Designs und der Entwicklung vertraut zu machen. Dieses breite Feld ist fast eine Wissenschaft für sich, über das es eine Menge Abhandlungen gibt – die meisten sind sehr trocken und theoretisch und in der Praxis nicht zu gebrauchen. Die wichtigsten Themen haben wir in den folgenden Abschnitten zusammengefasst, in denen wir die grundlegenden Fragen klären.

1.3.1 Auswählen von Namen

Variablennamen auszuwählen ist die Aufgabe, die Programmierer am häufigsten tun, aber am wenigsten reflektieren. Mit der Anzahl an verschiedenen Variablennamen, die in größeren Projekten auftauchen können, könnten Sie, wenn Sie den Namen, den Typ und die Deklaration jeder Variablen auflisteten, ein kleines »Telefonbuch« erstellen. Wie soll Ihr Verzeichnis aussehen? Im Laufe der Zeit haben sich verschiedene Namensschemata etabliert, hinter denen jeweils eine eigene Philosophie steht. Jedes hat seine Vor- und Nachteile. Prinzipiell lassen sie sich in zwei Gruppen unterteilen: kurze und einfache Variablen- und Funktionsnamen oder »sprechende« Variablen- und Funktionsnamen (das sind längere Namen, die Auskunft über den Typ und den Zweck geben).

Das »Telefonbuch« könnte folgendermaßen aussehen:

Name	Adresse	Nummer
J. D.	382 W. S.	-3951
M. S.	204 E. R.	-8382

Sehr informativ. Nun wissen Sie, dass Ihr Telefonbuch zwei Einträge enthält, aber auch nicht viel mehr. Sie kennen die Initialen der Person, aber nicht ihren vollständigen Namen. Sie wissen die Hausnummer, aber nicht den genauen Straßennamen. Und Sie sehen nur einen Teil der Telefonnummer.

Betrachten wir ein anderes Beispiel:

Name	Adresse	Nummer
ht5ft9in_age32_John Doe_male_married	386 West Street, Los Angeles, California, USA, Earth	+1-555-304-3951
ht5ft6in_age27_Mary Smith_female_single	204 East Road, Los Angeles, California, USA, Earth	+1-555-306-8382

In diesem Beispiel beinhaltet der Name der »Personen« die Größe, das Alter, den Genus und den Familienstatus. Die Adresse besteht nicht nur aus der Angabe der Strasse und der Stadt, sondern auch des Bundesstaates, des Landes und sogar des Planeten. Die Telefonnummer enthält zusätzlich den Länder- und Bereichscode.

Ist der zweite Ansatz besser als der erste? Keiner von beiden ist optimal. In den Vorlesungen zur Programmierung werden beide gelehrt, aber eigentlich ist keiner richtig zufriedenstellend. Für eine einfache `for()`-Schleife einen Typ `tpIntMyIntegerCounter` zu definieren und dann eine Variable `instMyIntegerCounterInstance` zu deklarieren, scheint ein bisschen übertrieben, wenn Sie nur ein Array durchsuchen und alle Elemente auf Null setzen müssen (siehe Listing 1.1).

```
for ( $instMyIntegerCounterInstance = 0;
      $instMyIntegerCounterInstance < MAXTPINTEGERCOUNTERRANGE;
      $instMyIntegerCounterInstance++)
         $instMyArrayInstance[$instMyCounterInstance] = 0;
```

Listing 1.1: Eine »Überdosis« Exaktheit

Auf der anderen Seite sind bloße Indizes wie `i`, `j`, `k` (anstelle der Langformen `$instMyIntegerCounterInstance`) genauso inakzeptabel, wenn Sie komplexe Zwischenspeicheroperationen wie Komprimierung o. ä. durchführen müssen.

Dies ist nur ein Beispiel für den Missbrauch eines allgemeinen Konzepts. Was ist also zu tun? Die Lösung besteht darin, ein gutes, allumfassendes Konzept zu wählen und an den richtigen Stellen Ausnahmen zu machen. Wenn Sie ein

Programm schreiben, wissen Sie genau, was an welcher Stelle Ihres Programms passiert und können schnell von einer zu einer anderen Stelle springen. Anderen fällt dies möglicherweise nicht so leicht. Wenn Sie eine Quelldatei von jemand anderem aus Ihrem Team erhalten und ihr eine Reihe von Funktionen hinzufügen wollen, müssen Sie sich zunächst einen Überblick verschaffen und die verschiedenen Abschnitte des Programms erkennen. Idealerweise geschieht dies, während Sie den Code lesen. Dies geht aber nur, wenn Sie den Quellcode mithilfe von Hinweisen und bekannten Mustern strukturieren. Deshalb sollten Sie soviel zusätzliche Informationen wie möglich in den Quellcode packen, ohne die eigentlichen Daten zu überfrachten. Wie können Sie diese Informationen also erhalten und sie in Ihren eigenen Code integrieren?

- Machen Sie Ihren Code leicht lesbar.
- Fügen Sie, wo möglich, Kommentare hinzu.
- Wählen Sie sprechende Variablennamen, wenn es passt.
- Sorgen Sie für klare und konsistente Funktionsschnittstellen.
- Unterteilen Sie Ihren Code in logische Funktionsgruppen.
- Trennen Sie separate Teile des Codes ab.
- Verwenden Sie Dateien, um Ihre Funktionen nicht nur logisch, sondern auch physikalisch in Gruppen zu unterteilen.
- Erstellen Sie eine Dokumentation.

Auf die einzelnen Punkte werden wir in den folgenden Abschnitten näher eingehen.

1.3.2 Lesbarmachen Ihres Codes

Um den Text während des Lesens zu verstehen, muss Ihr Gehirn die Informationen analysieren, die es von den Augen erhält, die wichtigen Teile identifizieren und dann diese Teile in die richtige Reihenfolge bringen. Die Analyse geschieht in zwei Schritten: die physikalische und die logische Analyse. Zunächst wird die physikalische Analyse durchgeführt, indem die optische Struktur des Textes untersucht wird, z.B. Absätze, Zeilen, Spalten und Zwischenräume zwischen den Wörtern. Durch diesen Vorgang wird die Wahrnehmung des Textes als Ganzes (etwa dem Blatt Papier oder dem Computerbildschirm) in eine Wahrnehmung als baumartige Struktur kleinerer Einheiten umgewandelt. Wenn wir von einer Baumstruktur mit Stammknoten oben und Blättern unten ausgehen, enthält der obere Teil die generischen Informationen, z.B. die Reihenfolge der Absätze, die Sie lesen müssen. Am unteren Ende steht so etwas wie die Reihenfolge der Wörter in einer Zeile oder gar die Reihenfolge der Zeichen in einem Wort.

Bei der logischen Analyse wird diese physikalische Information gelesen, von oben nach unten abgearbeitet und in ein möglichst sinnvolles Ergebnis umgewandelt. Ob dies nun eine grammatikalische Übersetzung ist (welche Struktur hat der Satz) oder eine kontextuelle Übersetzung (was bedeutet dieser Satz), ist für die folgende Diskussion unerheblich; wichtig ist, dass die logische Analyse umso einfacher und schneller geht und seine Ergebnisse umso besser sind, je besser die Ergebnisse der physikalischen Analyse sind.

Die logische Analyse kann fehlende Informationen aus der physikalischen Analyse kompensieren, wenn auch nur in begrenztem Maße.

```
NehmenSiezumBeispieldiesenSatzwennSieihnlesenkönnenfunktioniertIhrlogischer
Analysatorsehrgut.
```

Wahrscheinlich können Sie obigen Satz lesen, aber Sie brauchen länger und müssen sich stärker konzentrieren als bei den übrigen Sätzen in diesem Buch. Für den ersten Schritt der Analyse fehlen wichtige Informationen (die Leerzeichen), und das sind Sie nicht gewohnt.

Wir können ihn vereinfachen, indem wir einige Satzzeichen hinzufügen:

```
NehmenSiezumBeispieldiesenSatz--
wennSieihnlesenkönnen,funktioniertIhrlogischerAnalysatorsehrgut.
```

Für den physikalischen Analysator sind die Satzzeichen nützliche Informationen. Beachten Sie, dass sich diese Version sehr viel leichter lesen lässt und Sie einfacher eine beliebige Stelle fokussieren können. Zum nächsten Schritt:

```
Nehmen Sie zum Beispiel diesen Satz -- wenn Sie ihn lesen können,
funktioniert Ihr logischer Analysator sehr gut.
```

Dies ist die übliche Leseweise für einen Satz, so wie Sie ihn auf natürliche Weise wahrnehmen. Wir können den Satz noch weiter strukturieren:

```
Nehmen Sie zum Beispiel
diesen Satz --
wenn Sie ihn lesen können,
funktioniert Ihr logischer Analysator
sehr gut.
```

Dies ist ein Beispiel für die extreme Verwendung physikalischer Mittel, die helfen, den Satz so schnell wie möglich zu verstehen. Die Trennung verhindert hier eigentlich den natürlichen Lesefluss, da Sie es nicht gewohnt sind, dass ein Satz in syntaktische Einheiten unterteilt wird. Beim Quellcode ist dies jedoch von Vorteil, da dieser häufig komplizierte Konstrukte, Formeln und ähnliches enthält. Hier erhält der Benutzer durch die klare physikalische Struktur des Quellcode eine wertvolle Unterstützung. Mittel zur Verdeutlichung können Einzug und Verwendung von Schlüsselbegriffen an exponierten Stellen sein.

Betrachten wir ein kurzes PHP-Programm:

```
<?function myfunc($myvar){$somevar=$myvar*2;return($somevar+1);}print
myfunc(1);?>
```

Der eigentliche Programmcode ist wahrscheinlich kein Meisterwerk, aber uns interessiert im Moment nur die Struktur. Wenn Sie den vorherigen Abschnitt nicht gelesen hätten, wären Sie in der Lage, sofort auf den Beginn der Hauptroutine zu zeigen? Könnten Sie die erste und letzte Anweisung der Funktion darin identifizieren? Auch wenn Sie die gewünschten Stellen schnell finden, beginnen Ihre Augen unvermeidbar am Anfang der Zeile und »scannen« den Programmcode von links nach rechts und stoppen, wenn Sie glauben, dass Sie das Ziel gefunden haben. Unbewusst liest Ihr Gehirn die gesamte Zeile noch einmal, weil ihm Informationen aus der physikalischen Analyse fehlen. Zur Kompensation der fehlenden Information aus dem ersten Schritt übernimmt Ihr logischer Analysator diesen Schritt und ist damit doppelt belastet. Genau wie bei einem Computer hat Ihr Gehirn nur eine begrenzte Kapazität. Die zusätzliche Belastung des logischen Analysators entspricht also dem Kapazitätsengpass des Gehirns, wenn dieses versucht, den Quellcode zu verstehen und zu behalten. Verstehen und behalten ist aber genau das, was andere Leute beim Lesen Ihres Quellcodes tun sollen und was Sie beim Lesen fremder Codes beabsichtigen.

Dies war eine nahezu wissenschaftliche Erklärung, warum die Formatierung von Quellcode sinnvoll ist. Gibt es noch einen anderen Grund? Ja! Gut formatierter Quellcode sieht einfach gut aus.

Im Folgenden finden Sie einige Richtlinien, die wir für die optimalen Mittel zur Formatierung des Quellcodes halten. Sie sind nicht bindend, aber sie werden allgemein als guter Stil betrachtet. Viele industrielle und Open Source-Projekte wurden auf diese Weise formatiert, und häufig zahlt es sich aus, in diesem Stil zu schreiben.

- Setzen Sie Block-Tags (`<?`, `?>`, `<?php`, `<%`, `%>`, `{`, `}` usw.) in separate Zeilen.
- Ziehen Sie Blöcke stets mit Tabulatoren ein (im Idealfall setzen Sie den Tabulator auf mindestens 4).
- Fügen Sie zwischen Schlüsselwörtern und Schlüsselzeichen Leerzeichen ein, insbesondere wenn Sie Berechnungen vornehmen.
- Fassen Sie logische Programmteile in einem Block zusammen, indem Sie diese in aufeinanderfolgende Zeilen platzieren und fügen Sie zwischen allen anderen eine Leerzeile ein.
- Trennen Sie zwei Blöcke durch eine Leerzeile.
- Setzen Sie Funktions-Header und -Footer mittels einer Leerzeile vom übrigen Text ab (der Import von globalen Variablen wird als Teil des Funktions-Header betrachtet).

- Integrieren Sie Blockkommentare in den Programmcode und verwenden Sie dabei denselben Einzug wie der Codeblock, zu dem der Kommentar gehört.
- Setzen Sie alle Zeilenkommentare zu einem Block in dieselbe Spalte.

Als Beispiel zeigt Listing 1.2 den obigen Programmteil neu formatiert.

```
<?

function myfunc($myvar)
{

    $somevar = $myvar * 2;

    return($somevar + 1);

}

print(myfunc(1));

?>
```

Listing 1.2: Neu formatierter Programmteil

Wie Sie feststellen werden, können Sie durch diesen Codeteil leichter navigieren.

Die Verwendung von Leerzeichen lässt sich noch weiter ausdehnen, indem Klammern und Schlüsselwörter ebenfalls durch Leerzeichen getrennt werden.

```
<?

function myfunc ( $myvar )
{

    $somevar = $myvar * 2;

    return ( $somevar + 1 );

}

print ( myfunc ( 1 ) );

?>
```

Hier scheint es etwas übertrieben, aber stellen Sie sich vor, dieser Code sei in Tausend weitere Programmzeilen integriert. Vielleicht ändern Sie dann Ihre Meinung. Einige Leute meinen, Leerzeichen zwischen Klammern seien eher

störend und irritierend als hilfreich beim Strukturieren des Textes. Zugegeben, manchmal stimmt dies tatsächlich. In den Beispielen dieses Buches wird dieses Formatierungsmittel nicht immer eingesetzt. Wir überlassen es Ihnen, ob Sie dieses Methode verwenden. Am wichtigsten ist, dass Sie konsequent sind. Wenn Sie sich für einen bestimmten Stil entschieden haben, behalten Sie ihn zumindest für das aktuelle Projekt bei. Wenn Sie Änderungen an Quellcodes von anderen vornehmen, versuchen Sie ebenfalls, ihren Stil einzuhalten. Konsistenz ist eine der wichtigsten Aspekte in der Entwicklung.

Lesen Sie alle Beispiel-Codes aufmerksam und versuchen Sie, den Stil zu kopieren. Passen Sie dabei Ihren Stil soweit an, bis Sie dem Original recht nahe gekommen sind. Sobald Sie damit zurechtkommen, werden Sie merken, dass die Mühe nicht umsonst war.

Bevor Sie weiterlesen, hier zur Motivation zwei Beispiele.

Der Programmcode in Abbildung 1.1 soll eine SQL-Anweisung erzeugen. Mit Ausnahme der letzten Zeile, in der eine Zeichenkette mit dem Inhalt »select *« einer Variablen namens `$query` zugeordnet wird, weist wohl nichts in Abbildung 1.1 auf den Zweck des Codes hin. Beim Code aus Abbildung 1.2 sehen Sie leicht, was passiert.

UltraEdit-32 - [C:\temp\heavy.php3*]

File Edit Search Project View Format Column Macro Advanced Window Help

heavy.php3* | phpIRC.php3

```
if ((!$cselect) and (!$category) and (!$qsearch) and (!$country)) {
  echo "<p>$nbsp<ul><ul><ul><ul><h1>Insufficient Input Data!</h1> Please
provide more Information";
  exit();
 }
        if (empty($cselect) && $category != "JOBS WANTED")
           $select .= " and category <> 'JOBS WANTED' ";
 if ($category) {
  $select.=" and category='".$category."'";
 } elseif ($cselect) {
  $select.=" and (".$cselect.")";
 }
 $filter="where  (recruiters.active='yes') and (jobs.expiry > now())
and description <> '' $select ";
     } else {
 $filter = stripslashes($filter);
     }
 $query = "select * , unix_timestamp(expiry) as nx_inserted from jobs left
join recruiters on jobs.recruiterID=recruiters.recruiterID $filter order by
inserted desc";
```

For Help, pre | Ln 29, Col. 1 | DOS | Mod: 04.10.99 15:28:14 | File Size: 779 | INS

Abbildung 1.1: Schlechter Code

```
UltraEdit-32 - [C:\phpIRC\phpIRC_0.9b\phpIRC.php3]
File Edit Search Project View Format Column Macro Advanced Window Help
heavy.php3  phpIRC.php3

        // open a socket
        $irc_connection_handle = fsockopen($server_name, $port_nr);

        // return error if connection failed
        if(!$irc_connection_handle)
                return(0);

        // ok, we have a connection - now set up

        // send USER command, we don't expect backtalk on that one
        fputs($irc_connection_handle, "USER $irc_identd $irc_host $server_na

        // now send NICK command, we might get backtalk on this one
        fputs($irc_connection_handle, "NICK $irc_nick\n\r");

        // fake connection status so that irc_get_message() works
        $irc_connected = 1;

        // now see if we got an erroneous nick message,
        // if yes, our nick is not correct
        $done = 0;
        while(!$done)
        {
                $msg = irc_get_message();

For Help, pre: Ln 547, Col. 1   DOS   Mod: 02.09.99 21:47:00   File Size: 61817   INS
```

Abbildung 1.2: Besserer Code

So, oder wenigstens ungefähr so, sollte unserer Meinung nach Programmcode aussehen. Der Code ist klar strukturiert, gut kommentiert und leicht zu verstehen.

1.3.3 Hinzufügen von Kommentaren

Wir können es nicht genug betonen – auch wenn Sie es während der Programmierung für das Überflüssigste von der Welt halten – Kommentierung ist wichtig, wenn Sie qualitativ hochwertigen Code produzieren. Bei der Lösung von komplizierten Problemen denken selten zwei Menschen auf dieselbe Weise. Was dem Einen vollkommen klar erscheint, kann für den Anderen unklar sein. In diesen Situationen sind Kommentare sehr hilfreich. Sie sollten daher, wenn möglich, immer eingefügt werden.

Es gibt zwei Hauptarten von Kommentaren: Header-Kommentare (z.B. Kommentare im Datei-Header, Modul-Header oder Funktions-Header) und im Text eingebettete Kommentare. Header-Kommentare sollten zur Einführung verwendet werden, um den Leser über allgemeine Dinge in einer Datei zu informieren oder über den nächsten größeren Programmabschnitt. Eingebettete Kommentare sollten innerhalb von Funktionen verwendet werden, um zu erläutern, was eine bestimmte Programmzeile oder ein Block genau macht.

Die folgenden Abschnitte sollen Ihnen einen Eindruck vermitteln, wie diese Kommentare aussehen und welche Informationen sie enthalten können. Inzwischen werden diese Kommentare mit RAD(Rapid Application Development)-Werkzeugen oder anderen Autorenhilfen erstellt, aber da es zum Zeitpunkt der Bucherstellung keine ähnlichen Systeme für PHP gibt, sollten Sie die Kommentare trotz der zusätzlichen Belastung von Hand schreiben.

In den folgenden Abschnitten werden die Kommentartypen nach Abstraktionsgrad geordnet erörtert. Wir beginnen dabei mit dem abstraktesten Typ.

Aktualisierung von Kommentaren

Denken Sie daran, die Kommentare einzugeben, bevor oder während Sie die Module oder Funktionen bearbeiten, die sie beschreiben. Es ist sehr ärgerlich, eine Datei im Nachhinein noch einmal öffnen zu müssen, nur um die Kommentare einzufügen. Achten Sie darauf, wenn Sie später Funktionen verändern, auch die Kommentare entsprechend zu ändern. Wenn Sie beispielsweise globale Variablen hinzufügen oder entfernen, aktualisieren Sie auch ihren Verwendungszweck im Kommentarkopf. Das Gleiche gilt für Änderungen in der Reihenfolge von Parametern, Parametertypen usw.

Einsatz von Makros zur Beschleunigung der Kommentierung

Erstellen Sie mit Ihrem bevorzugten Editor für jeden Kommentartyp Makros und weisen Sie diesem Tastaturkürzel zu (z.B. `E`íêäF+E^äíF+EcNF` für Dateiköpfe, `E`íêäF+E^äíF+EcOF` für Modulköpfe usw.). Wenn der Editor es unterstützt, fügen Sie auch Variablen in diesen Kommentaren ein, so dass Sie ausgefeilte und aussagekräftige Kommentare über einen kurzen Frage-Antwort-Dialog erzeugen können.

Datei-Header-Kommentare

Datei-Header-Kommentare sollten in etwa so aussehen wie jene im Listing 1.3.

```
///////////////////////////////////////////////////////////////////////////
//
// phpIRC.php3 - IRC client module for PHP3 IRC clients
//
///////////////////////////////////////////////////////////////////////////
//
// This module will handle all major interfacing with the IRC server,
// making all IRC commands easily available by a predefined API.
//
// See phpIRC.inc.php3 for configuration options.
//
// Author: Till Gerken  Last modified: 09/17/99
//
```

```
// Copyright (c) 1999 by Till Gerken
//
/////////////////////////////////////////////////////////////////////////////
```

Listing 1.3: Datei-Header-Kommentar

Möglicherweise möchten Sie lieber einen umrandeten Kasten verwenden, der von mehrzeiligen Kommentaren erzeugt wird. Manche Leute finden ihn ästhetischer (siehe Listing 1.4).

```
*****************************************************************************
*                                                                           *
* phpIRC.php3 - IRC client module for PHP3 IRC clients                      *
*                                                                           *
*****************************************************************************
*                                                                           *
* This module will handle all major interfacing with the IRC server, making *
* all IRC commands easily available by a predefined API.                    *
*                                                                           *
* See phpIRC.inc.php3 for configuration options.                            *
*                                                                           *
* Author: Till Gerken  Last modified: 09/17/99                              *
*                                                                           *
* Copyright (c) 1999 by Till Gerken                                         *
*                                                                           *
*****************************************************************************
```

Listing 1.4: Datei-Header-Kommentar mit mehrzeiligen Kommentaren

Extraktion von Blockkommentaren in UNIX

Auf UNIX-Systemen werden Blockkommentare mit dem Befehl `grep` aus der Quelle gelesen:

```
grep '^[\\\/ ]*\*' source.php3
```

Welchen Stil Sie zur Formatierung Ihres Kopfes wählen, ist unwesentlich; vielmehr zählt, welche Informationen Sie in den Datei-Header einfügen. Wie in diesem Beispiel gezeigt, sollten im Kopf allgemeine Informationen über das Modul, den Autor usw. enthalten sein. Die Punkte sollten in sinnvoller Reihenfolge aufgeführt werden (z.B. macht es keinen Sinn, erst eine lange Beschreibung und dann eine Kurzbeschreibung zu geben – nachdem Sie die lange gelesen haben, brauchen Sie die kurze nicht mehr). Die folgende Liste zeigt Empfehlungen (nach Wichtigkeit geordnet):

1. Dateiname des Moduls
2. Kurzbeschreibung des Moduls (eine Zeile)
3. Ausführliche Beschreibung des Moduls

4. Angaben über Verwendung, Anforderungen, Warnungen usw.
5. Name des Autors und Kontaktadresse
6. Erstellungsdatum des Moduls und Datum der letzten Änderung
7. Urheberrechtsvermerk
8. Lizenzinformation
9. Verweise auf Änderungsprotokoll, Home Page, Distributionsdatei usw.
10. Schließlich Auszüge aus dem Änderungsprotokoll, wenn erforderlich.

Wenn dies nach zu viel Informationen aussieht, bedenken Sie, dass Sie besser redundante Informationen haben, als wenn Informationen fehlen. Natürlich sind nicht alle Felder in allen Situationen geeignet; auch in den vorhergehenden Beispielen haben wir nicht alle Felder aufgenommen. Trotzdem sollten Sie so viel Informationen wie möglich in den Kopf packen. Das Schlimmste, was passieren kann, ist, dass einige Leute sie nicht lesen. Andere sind vielleicht dankbar dafür. Vielleicht sogar Sie, denn das Weglassen von Urheberrechts- und Lizenzinformationen in einem kommerziellen Produkt kann später einige Kopfschmerzen bereiten, wenn andere Programmierer Ihren Code zur freien Verwendung weiterverwenden.

Modul-Header-Kommentare

Wenn Ihre Datei mehr als ein Modul hat (z.B. wenn ein Modul nur aus drei Funktionen besteht, welche die Funktionalität aus einer größeren Prozedur eines übergeordneten Moduls extrahieren), sollten Sie vor der ersten Funktion einen Informationsblock einfügen.

Ein Modul-Header sollte in etwa so aussehen wie in Listing 1.5.

```
///////////////////////////////////////////////////////////////////////////
//
// Submodule for file access from main()
//
///////////////////////////////////////////////////////////////////////////
//
// This submodule will provide functionality for easy file access,
// and includes error checking and reporting.
//
// Functions:
//
//     int file_open(string $file_name)
//     bool file_close(int $file_handle)
//     int file_read(int $file_handle, $nr_bytes)
//
// Remarks:
//
```

```
//     - provides no seek function
//     - does not allow write access
//
///////////////////////////////////////////////////////////////////////////
```

Listing 1.5: Modul-Header-Kommentar

Ein solcher Header könnte folgende Informationen enthalten (nach Wichtigkeit geordnet):

1. Kurzbeschreibung des Moduls
2. Ausführliche Beschreibung des Moduls
3. Funktionsprototypliste
4. Anmerkungen / Hinweise

Auch hier können die Kommentare mehrzeilig sein.

Funktions-Header-Kommentare

In den Funktions-Header-Kommentaren sollten so detailliert wie möglich für jede Funktion Syntax, Zweck und notwendige Aufrufdaten angegeben werden (siehe Listing 1.6). Diese Art des Kommentars ist in eingebetteten Kommentaren nicht so wichtig. Funktions-Header-Kommentare dienen dazu, den Programmierer während der Entwicklung und Erweiterung eines Moduls rasch über die Anforderungen und Eigenarten der einzelnen Funktionen zu informieren. Diese Informationen benötigen hauptsächlich fremde Entwickler, welche die ursprünglichen Funktionen nicht erstellt haben. Sind solche Kommentare nicht vorhanden, muss sich der Entwickler selbst ins Programm vertiefen, um die gewünschte Information zu finden. Dies führt häufig zu Fehlern, da nicht alle versteckten Fallen (die manchmal wirklich nicht leicht zu finden sind) gefunden werden.

```
///////////////////////////////////////////////////////////////////////////
//
// int irc_get_channel_by_name(string $name)
//
///////////////////////////////////////////////////////////////////////////
//
// Searches a channel by its name in the internal channel table and returns
// its handle.
//
///////////////////////////////////////////////////////////////////////////
//
// Parameter:
//     $name - name of channel to search for
//
```

```
//////////////////////////////////////////////////////////////////////////
//
// Return value:
//    Channel handle (numeric), 0 on error
//
//////////////////////////////////////////////////////////////////////////
//
// Global references:
//    $irc_channel_array
//
//////////////////////////////////////////////////////////////////////////
```

Listing 1.6: Typischer Funktions-Header-Kommentar

Ein Funktions-Header-Kommentar sollte einige der folgende Punkte enthalten (in geordneter Reihenfolge):

1. Funktionsprototyp
2. Detaillierte Beschreibung der Funktionen
3. Anmerkungen / Hinweise
4. Parameterbeschreibung
5. Beschreibung des Rückgabewertes
6. Globale Verweise
7. Autor und Datum der letzten Änderung

Eingebettete Kommentare

Eingebettete Kommentare werden direkt im Programm eingefügt. Sie sollen die Fragen beantworten, die an dieser Stelle auftreten. Während Sie programmieren, ist Ihnen natürlich alles ganz klar. Das ist meist die Ursache dafür, dass zu wenige Kommentare eingefügt werden. Wenn Sie die Datei später erneut öffnen, vielleicht sogar erst nach einem Jahr, haben Sie möglicherweise einige der Strukturen vergessen und wissen nicht mehr, warum Sie diese benutzten. Das haben wir häufig selbst erlebt, sowohl in unseren Programmen als auch in jenen von anderen. Als goldene Regel merken Sie sich einfach, dass Sie kaum zu viele eingebettete Kommentare schreiben können. Die einzige Ausnahme hiervon ist, wenn zu viele Kommentare den Programmcode, den sie beschreiben sollen, zu unübersichtlich machen. Offensichtliche Dinge sollten Sie nicht kommentieren. Ein paar Beispiele finden Sie in Listing 1.7.

```
function calculate_next_index ( $base_index )
{

    $base_index = $base_index + 1;           // increase $base_index by one
```

```
    //
    //''''
    // Table of contents
    //
    // 1. Introduction
    // 2. About the authors
    [LOTS of lines cut out]
    //
    //
    $new_index = $base_index * COMPLICATED_NUMBER / 3.14 + sin($base_
            ➥index);

}
```

Listing 1.7: Schlechte eingebettete Kommentare

In der ersten Zeile wird `$base_index` um 1 erhöht – muss dass kommentiert werden? Wir bezweifeln es. Jeder kann sehen, dass `$base_index` erhöht wird. Aber warum wird es erhöht und warum gerade um 1? Ein besserer Kommentar wäre »Jump to the next ordinal index we want to point to; it's exactly one element away« .(Springe zum Index mit der nächsten Ordnungszahl, er ist genau ein Element weit entfernt).

Dieselbe Art von Problem stellt sich auch beim zweiten Kommentar, allerdings aus einem anderen Grund. Der Programmierer hat die vollständige Referenz für den Algorithmus eingefügt, darunter leider auch Müll. Natürlich ist es gut, ausführlich zu beschreiben, was man tut, aber Sie müssen unterscheiden, was wichtig ist und was nicht.

Wenn Sie Programme kommentieren, stellen Sie sich selbst folgende Fragen:

- Was tun Sie gerade?
- Warum tun Sie es?
- Warum tun Sie es auf diese Weise?
- Warum tun Sie es an dieser Stelle?
- Wie wirkt sich dieser Programmteil auf das übrige Programm aus?
- Was benötigt dieser Programmteil?
- Hat dieses Verfahren Nachteile?

Wenn Sie beispielsweise Zeichenketten analysieren, dokumentieren Sie das Format der eingegebenen Zeichenketten, die Toleranzen Ihres Parsers (seine Reaktion auf Fehler und falsche Eingaben) und die Ausgabe. Wenn dies alles zu viele Informationen sind, um sie direkt im Programm einzufügen, hinterlassen Sie zumindest einen Verweis auf die externe Dokumentation, in wel-

cher sich der Leser über alle Aspekte des Parsers informieren kann. Denken Sie auch daran, Ihren Funktions-Header zu aktualisieren, indem Sie dort einen Verweis auf die Dokumentation platzieren.

1.3.4 Auswählen von sprechenden Namen

Wie bereits erwähnt ist die Auswahl eines geeigneten Namen für eine Variable oder eine Funktion ein wichtiger Punkt in der Programmierung. Ganz allgemein sollten Sie bei der Namensgebung zunächst berücksichtigen, ob es sich um eine lokale oder globale Variable handelt. Wenn die Variable nur im lokalen Umfeld einer Funktion auftaucht, wählen Sie einen kurzen, präzisen Namen, der den Inhalt oder die Bedeutung dieser Variablen wiedergibt. Der Variablenname sollte aus maximal zwei Wörtern bestehen, die durch einen Unterstrich oder Großbuchstaben voneinander getrennt sind. Betrachten Sie hierzu Listing 1.8.

```
$counter
$next_index
$nrOptions
$cookieName
```

Listing 1.8: Beispiele für Namen lokaler Variablen

Achten Sie darauf, Namensschemata nicht zu vermischen! Verwenden Sie entweder nur Kleinbuchstaben und trennen Sie die Wörter durch einen Unterstrich, oder benutzen Sie Großbuchstaben zur Trennung. Natürlich können Sie auch Großbuchstaben und Unterstrich verwenden, aber verwenden Sie nicht Großbuchstaben für eine Variable und Unterstrich für eine andere. Dies führt zu Fehlern und zeugt von schlechtem Stil. Sobald Sie Ihren eigenen Stil gefunden haben, behalten Sie ihn für das gesamte Projekt konsequent bei.

Alle globalen Variablen sollten ein Präfix haben, welches das Modul kennzeichnet, zu dem es gehört. Dieses Vorgehen erleichtert die Zuordnung der globalen Variablen zu ihren Modulen und hilft, Konflikte zu vermeiden, wenn zwei Variablen mit demselben Namen aus verschiedenen Modulen im gleichen globalen Umfeld vorkommen. Das Präfix sollte durch Unterstrich vom eigentlichen Variablennamen getrennt sein. Es sollte aus einem einzigen Wort bestehen, am besten einer Abkürzung (siehe Listing 1.9).

```
$phpPolls_setCookies
$phpPolls_lastIP
$db_session_id
$freakmod_last_known_user
```

Listing 1.9: Beispiele für Namen globaler Variablen

Der Vorteil von kleinen Größen

Erzeugen Sie lieber mehrere kleinere Projekte, in denen Sie jeweils ein anderes Namensschema verwenden. Dies hat folgende Vorteile:

- Sie haben die Möglichkeit, Ihren ganz eigenen Stil zu finden.
- Wenn Sie den Stil eines Anderen übernehmen müssen, haben Sie bereits Erfahrung.

Wie diese Beispiele zeigen, sind die Namen globaler Variablen in der Regel länger als die Namen lokaler Variablen. Dies liegt nicht nur an dem Modulpräfix, sondern auch an der exakteren Abgrenzung. Wenn nicht klar erkennbar ist, an welchen Stellen eine Variable definiert und initialisiert wurde, weil diese in einem Modul liegen, auf das Sie keinen Zugriff haben, sollten Sie im Namen unbedingt den Inhalt und die Bedeutung der Variablen angeben. Natürlich geht dies nicht unbegrenzt – wer will sich schon 40 Zeichen merken – aber das ist mehr eine Grenze, die der gesunde Menschenverstand setzt.

Im Prinzip sollten Sie globale Variablen so benennen, als würden Sie diese jemandem beschreiben. Wie würden Sie zum Beispiel die Variable `$phpPolls_lastIP` beschreiben? Möglicherweise wissen Sie nicht, was »phpPolls« macht, aber der Name legt nahe, dass es etwas mit »Pollen« zu tun hat. `lastIP` besagt, dass es sich um die letzte IP handelt ... Welche IP? Das wissen Sie nicht. Offensichtlich ist der Name für diese Variable nicht sehr glücklich gewählt, weil er den Inhalt nicht genau erklärt. Angenommen, Sie fragten nach dem Zweck dieser Variablen und erhielten die Antwort: »Sie enthält die IP des letzten Benutzers, der zuletzt angerufen hat. Jetzt denken Sie sich einen Namen aus. Wie wäre es mit `$phpPolls_last_voters_IP`? Besser, oder? Aber auch wenn der Name selbst jetzt in Ordnung ist, passt er nicht, weil es noch zwei weitere globale Variablen aus der Reihe »phpPolls« gibt, die jeweils mit dem Präfix `phpPolls_` beginnen, während der Rest in einem Wort geschrieben wird. Aus Konsistenzgründen sollten Sie die Wörter innerhalb des Namens nur durch Großbuchstaben trennen: `$phpPolls_lastVotersIP`.

Funktionsnamen sollten mit genauso soviel Sorgfalt ausgewählt werden wie Namen globaler Variablen, wenn auch mit ein paar Änderungen. Funktionsnamen sollten Auskunft über ihre Funktion geben und sich in den Lesefluss einpassen. Diese Integration erreichen Sie, indem Sie die Aktion ermitteln, die eine Funktion durchführt, und einen in den meisten Fällen akzeptablen Namen für diese Aktion wählen.

Wenn eine Funktion feststellt, ob ein Benutzer gerade online ist, könnte der Name dieser Funktion z.B. folgendermaßen aussehen:

```
function get_online_status($user_name);
function check_online_status($user_name);
function user_status($user_name);
```

```
function user_online($user_name);
function is_user_online($user_name);
```

Je nach Rückgabetyp ist nur der erste oder letzte Name dieser Liste passend. Angenommen, diese Funktion gäbe einen booleschen Wert zurück, dann würde sie in der Regel in Zusammenhang mit einer `if()`-Klausel benutzt. Dies sähe in etwa so aus:

- Alternative 1:

```
if(user_status($user_name))
{
    // do something
}
```

- Alternative 2:

```
if(is_user_online($user_name))
{
    // do something
}
```

Im ersten Fall sieht der Funktionsname etwas deplaziert aus: »Wenn der Benutzerstatus von John, dann tue etwas«. Vergleichen Sie dies mit der zweiten Alternative: »Wenn Benutzer John online ist, dann tue etwas«. Die zweite Variante unterbricht nicht den Lesefluss und erscheint auf den ersten Blick sinnvoller. Die erste Variante lässt Fragen offen: Auf welchen Status wird Bezug genommen und wie wird der Status ausgegeben? Der zweite Funktionsname sagt klar, dass diese Funktion den Online-Status von jemanden überprüft und das Ergebnis als boolescher Wert zurückgibt.

Was wäre, wenn das Ergebnis in einem variablen Parameter dieser Funktion zurückgegeben würde?

- Alternative 1:

```
function user_status($user_name, &$status)
{
    // retrieve status and return in $status
}

$success = user_status($user_name, $user_online);
```

- Alternative 2:

```
function get_online_status($user_name, &$status)
{
    // retrieve status and return in $status
}

$success = get_online_status($user_name, $user_online);
```

`user_status()` ist zwar keine schlechte Wahl für einen Namen mit diesem Zweck, aber `get_online_status()` passt besser. Das Wort `get` verdeutlicht, dass die Funktion den Online-Status einliest und ihn irgendwo speichert – entweder in einer globalen Variablen oder in einem variablen Funktionsargument.

Verwenden Sie für Funktionen, die nur Daten verarbeiten, anstelle von passiven Namen nur aktive Namen. Benutzen Sie keine Hauptwörter, wie `huffman_encoder()` oder `database_checker()` – nennen Sie die Funktionen `huffman_encode()` und `check_database()` oder kehren Sie die Reihenfolge der Wörter um, wenn dies für Ihr Modulpräfix besser passt.

Ist Ihr Programmcode zweisprachig? Dreisprachig?
Eine der häufigsten Kritiken an Programmcodes betrifft die »Verstaatlichung«, die Durchsetzung der Programmiersprache (die normalerweise einen englischsprachigen Ursprung hat) mit einer anderen Sprache. In unserem Fall (Tobias ist Italiener, Till ist Deutscher) haben wir bei der Durchsicht von Projekten lokaler Programmierer festgestellt, dass diese gerne deutsche bzw. italienische Variablen- und Funktionsnamen anstatt englischer Namen verwendeten. Dies führte zu einer seltsamen Mischung. In Ihren üblichen Briefwechseln verwenden Sie vermutlich keine Mischung aus Englisch, Französisch, Spanisch oder Ähnliches. Warum sind Sie dann nicht auch bei der Programmierung konsequent und verwenden in PHP englische Namen? Dies hilft auch anderen zu verstehen, was Sie geschrieben haben.

1.3.5 Generieren von klaren und konsistenten Schnittstellen

Es geht Ihnen vielleicht auf die Nerven, schon wieder das Wort »konsequent« zu hören, aber für die Entwicklung von Schnittstellen ist Konsequenz das A und O.

Leider ist ein Negativbeispiel in PHP nicht greifbar.

Beim Auto befindet sich das Gaspedal auf der rechten und die Bremse auf der linken Seite. Wenn Sie sich ein neues Auto kaufen, erwarten Sie, dass es genau dieselbe Anordnung hat. Sie erwarten, dass eine rote Ampel »Stop« und eine grüne Ampel »Freie Fahrt« bedeutet. Genauso ist es bei der Datenverarbeitung. Angenommen, Sie benutzen eine Bibliothek, um auf Dateien zuzugreifen, und verwenden für jede Funktion eine andere Dateibearbeitungsroutine. Es wäre seltsam, wenn die Funktion, die zum Einlesen der Datei dient, die Routine als ersten Parameter erwartet, während die Schreibfunktion sie als letzten Parameter und die dritte Funktion sie irgendwo in der Mitte der Parameterliste erwartet.

Bei der Entwicklung einer Schnittstelle sollten Sie zunächst folgende Punkte überdenken:

- Welche Daten sollen über die Schnittstelle ausgetauscht werden?
- Welche Parameter brauche ich wirklich?
- Welche Parameter werden üblicherweise von den meisten (oder allen) Schnittstellenfunktionen benutzt?
- Welche Reihenfolge wäre für die Parameter logisch?

Sobald Sie sich für eine bestimmte Vorgehensweise entschieden haben, halten Sie diese konsequent durch; machen Sie in Ihrem Modul keine Ausnahme von der Regel. Selbst die internen Funktionen sollten dieser Regel entsprechen. Diese Strategie ermöglicht Ihnen später, interne Funktionen in der Schnittstelle zur Verfügung zu stellen. Außerdem werden es Ihnen auch Ihre Team-Mitglieder danken, wenn sie neuen Programmcode in Ihr Modul integrieren müssen.

Wenn Sie sich die Zeichenketten-Funktionen im PHP-Handbuch ansehen, finden Sie `strpos()`, `strchr()`, `strrchr()` usw. Alle diese Funktionen akzeptieren die Parameter `string haystack`, `string needle`, wobei `haystack` die Zeichenkette ist, in welcher gesucht wird, und `needle` die Zeichenkette ist, nach welcher gesucht wird. Sehen Sie sich nun `str_replace()` an. Diese Funktion führt nicht nur ein neues Namensschema ein, seine Argumente sind auch das genaue Gegenteil der übrigen Funktionen: sie akzeptiert `string needle`, `string haystack`.

Als wir nach dem Grund für diese Diskrepanz fragten, erhielten wir die Antwort, dass `str_replace()` der schnelle Ersatz für `ereg_replace()` ist und dass die meisten Leute ihre Aufrufe von `ereg_replace()` (bei welcher die umgekehrte Argumentenreihenfolge akzeptiert werden) in `str_replace()` änderten. Natürlich ist dieses Argument berechtigt. Aber warum akzeptieren die regulären Ausdrucksfunktionen ihre Argumente in einer der Zeichenketten-Funktionen entgegengesetzten Reihenfolge? Weil diese Funktionen in PHP jene in C reflektieren. Bei der Entwicklung eines Programms ist es ärgerlich zu sehen, dass sich `str_replace()` von dem Rest der Funktionsgruppe abhebt. Wenn Sie die Schnittstellen Ihrer nächsten Bibliotheken entwickeln, achten Sie darauf, dass diese Situation bei Ihnen nicht vorkommt.

1.3.6 Unterteilung des Programms in logische Gruppen

Programme bestehen in der Regel aus mehreren Funktionsgruppen, die sich jeweils einer speziellen Aufgabe und / oder einem Anwendungsbereich widmen. Beim Schreiben eines datenbankgestützten Programms sollte eine separate Funktionsgruppe für den Datenbankzugriff verantwortlich sein. Dieser Programmcode stellt eine eigene Einheit dar und kann sicher vom übrigen

Programm abgetrennt werden – sofern Sie es gut durchdacht haben. Funktionsgruppen, die logisch nur eine bestimmte Aufgabe durchführen können, sollten so konzipiert sein, dass sie vom übrigen Programm unabhängig behandelt werden können. Diese Funktionen sollten auch physikalisch vom restlichen Hauptprogramm getrennt sein, in Form eines Moduls.

Bevor Sie ein Programm installieren, sollten Sie eine Liste aller Funktionen erstellen, die in einem Modul zusammengefasst werden können und einen separaten Entwicklungsplan für jedes Modul erarbeiten. Erstellen Sie unbedingt detaillierte Datenflussdiagramme, damit die Module alle Anforderungen des Programms handhaben können. Unterschätzen Sie nicht die Bedeutung der Vorbereitung auf Papier. Es würde zu weit führen, wenn wir auf diesen Aspekt ausführlich eingingen. Wir empfehlen Ihnen, sich mit diesem Thema in einigen der hervorragenden Bücher über Design-Verfahren vertraut zu machen.

1.3.7 Abtrennen separater Programmteile

Ganze Blöcke aus dem Programm zu entfernen, sollte sowohl während der Planungs- als auch während der Installationsphase geschehen. Angenommen, eine Funktion führt folgende Aktionen durch:

1. Datei öffnen
2. Datenblock aus der Datei lesen
3. Daten validieren
4. Mögliche Datenfehler korrigieren
5. Daten wieder in die Datei zurückschreiben
6. Datei schließen

Jeder Schritt kann in einem separaten Block programmiert werden. Guter Stil ist, diese Blöcke zu separieren und sie in separate Funktionen umzuwandeln. Damit können Sie die einzelnen Blöcke nicht nur in anderen Funktionen wiederverwenden (vielleicht benötigen Sie die Unterstützung für die Dateioperationen auch an anderer Stelle); der Programmcode ist auch leichter zu lesen, und Fehler sind schneller zu finden. Sie können die entnommenen Teile absichern, sie mit separaten Fehlersuchroutinen ausstatten u. v. m. Wenn Sie dies alles direkt in das Programm einbetteten, würde dies schnell zu einer unhandlichen Größe aufgeblasen. Wenn Sie dieselben Codeblöcke in anderen Funktionen verwendeten, müssten Sie diese gegebenenfalls auch in allen anderen Funktionen manuell korrigieren, die diese Blöcke verwenden – ganz zu schweigen von Fehlern und Änderungen. Durch das Heraustrennen fassen Sie die kritischen Stellen zusammen, durch die Korrektur einer einzelnen Zeile können Sie das Verhalten aller verwandten Funktionen ändern.

1.4 Verwenden von Dateien zur Gruppierung von Funktionen

Wir haben gezeigt, dass es sich auszahlt, mehrere Dateien für den Quellcode zu verwenden. Wir empfehlen Ihnen, auch für die meisten anderen Ressourcen Dateien zu verwenden, wie Konfigurationsangaben, benutzerspezifische Header, Footer oder andere Vorlagen. Das Gleiche gilt für alles Andere, das Sie als separate Einheit von Ihrem Projekt abtrennen können.

Für ein und dasselbe Projekt mehrere Dateien zu verwenden, bietet einige Vorteile:

- Sie erhalten kleinere Quellcode-Dateien, die leichter zu pflegen sind.
- Sie können mehrere Versionen für jede Datei erzeugen, anstatt für jede kleine Änderung das gesamte Projekt durchsuchen zu müssen.
- Sie können Ressourcen vom Projekt abtrennen und sie in anderen Projekten wiederverwenden.
- Mehrere Team-Mitglieder können gleichzeitig an demselben Projekt arbeiten, ohne dass sie Gefahr laufen, sich in die Quere zu kommen, wenn Sie alle Dateien bei der Versionskontrolle überprüfen.

Diese Punkte gelten für die meisten Ressourcen, die in einem Projekt vorhanden sein können.

Die Dateien sollten einen aussagekräftigen Namen erhalten, der ihren Inhalt wiedergibt. Wenn mehrere Dateien zu einer größeren Gruppe gehören, sollte dem Namen möglicherweise ein Präfix vorangestellt werden. Legen Sie die Dateien in Unterverzeichnissen des Projekt-Wurzelverzeichnisses ab. Angenommen, Sie haben eine Datenbank-Abstraktionsebene mit Modulen, die auf verschiedene Datenbanken zugreifen, wobei diese in einzelnen Dateien gepackt sind. Hier sollten alle Dateien das Präfix dba_ tragen (wobei dba für Datenbank-Abstraktion steht), so dass ihre Dateien `dba_mysql`, `dba_odbc`, `dba_oracle` usw. heißen.

Stellen Sie sicher, dass Sie die Unterverzeichnisse später ändern können, indem Sie konfigurierbare Modulverzeichnisse verwenden. Ein Beispiel (beachten Sie, dass sich dba in diesem Beispiel nicht auf die PHP-Funktionen dba_* beziehen):

```
<?
require("config.php3");

require("$dba_root/dba.php3");
require("$socket_root/socket.php3");
require("$phpPolls_root/phpPollUI.php3");
```

```
// [...]
?>
```

Die Variablen `$dba_root`, `$socket_root` und `$phpPolls_root` in diesem Beispiel sollten in einer zentralen Konfigurationsdatei gespeichert sein und globale Optionen für das gesamte Projekt enthalten. Diese Konfigurationsdatei sollte nur diejenigen Optionen aufführen, die von allen Quelldateien unabhängig von den anderen benötigt werden und daher global zur Verfügung gestellt werden müssen. Zu diesen Optionen können Umgebungsoptionen gehören, wie z.B. der Name des Standorts, die Speicherorte des Dateisystems usw.

Bleiben Sie auf dem (allgemeinen) Pfad
Wenn Sie die Konfigurationsdatei aus einem Unterverzeichnis einfügen, verwenden Sie stets relative Pfade. So stellen Sie sicher, dass sich Ihr Projekt in Ihrem Dateisystem und den Systemen des Benutzers frei bewegen kann. Verlassen Sie sich nicht darauf, dass spezielle Bedingungen Ihrer Entwicklungsumgebung auch in allen eingesetzten Umgebungen vorhanden sind. Was immer Sie allgemein halten können, sollten Sie allgemein halten.

1.5 Schreiben einer Dokumentation

Neben der Kommentierung und Strukturierung ist die Dokumentation ein wichtiger Teil der Entwicklungsarbeit. Die Dokumentation eines Projektes ist möglicherweise der erste Teil des Projekts, den Ihr Kunde sieht – und der erste Eindruck zählt.

Professionell entwickelte Dokumentationen, die mehr als das obligatorische »Befolgen Sie die Installationsanweisungen in der README-Datei« enthalten, sollte Ihnen in Fleisch und Blut übergehen. So wie Sie ein gut geschriebenes Handbuch für Ihr Mobiltelefon, einen neuen Monitor oder andere technische Geräte erwarten, selbst wenn Sie diese in dem kleinsten Geschäft gekauft haben, erwarten Ihre Kunden eine gute Dokumentation von Ihnen (nicht zu vergessen, dass Sie möglicherweise eine Menge Geld dafür bezahlt haben).

Wie bei Kommentaren werden ausführliche Dokumentationen üblicherweise mithilfe von RAD-Programmen erstellt. Leider gibt es bisher noch keine Werkzeuge, die speziell auf PHP zugeschnitten sind. Die Handbuch-Erstellung bleibt daher eine nicht unterstützte und undankbare Aufgabe. Trotzdem ist sie notwendig. Sie sollte jedoch nicht Ihre Produktivität behindern. Ein vollständiges Handbuch sollte wie ein kleines Buch aufgebaut sein und folgende Punkte enthalten:

- Einführung
- Inhaltsverzeichnis
- Benutzer-Leitfaden
- Technische Dokumentation
- Entwickler-Leitfaden
- Vollständige Referenz aller Funktionen

Der Benutzer-Leitfaden sollte ausführlich alle Funktionen Ihrer Anwendungsprogrammschnittstelle (sofern vorhanden) für den Durchschnittsbenutzer beschreiben. Gehen Sie nicht zu sehr ins technische Detail; es sollte nur eine »Wie Sie ... tun«-Beschreibung sein. Behandeln Sie aber jeden Aspekt. Die technische Dokumentation sollte für technisch interessierte Benutzer und Administratoren geschrieben werden und Auskunft über die technischen Spezifikationen Ihres Programms, gebräuchliche und neue Standards sowie über interne Datenverarbeitung geben (sofern dies für den Leser interessant ist) – und soweit es Ihre Lizenz zulässt. Wenn Ihre Kunden den Quellcode ansehen und / oder ändern können, fügen Sie auch einen Entwickler-Leitfaden hinzu, in dem Sie die Projektstruktur, den Datenfluss und die internen Beziehungen aufführen. Außerdem sollten Sie der vollständigen Beschreibung eine Referenz aller Funktionen (einschließlich interner Funktionen) beifügen.

Wenn Sie in einem Team arbeiten, kann ein professioneller Technischer Redakteur eine echte Hilfe sein. Er hat Erfahrung darin, wie umfassende technische Dokumentationen erstellt werden, und auch die Zeit, dies zu tun. Wenn Sie diese Aufgabe einem Entwickler auftragen, der nebenbei noch programmieren soll, führt dies zu jeder Menge zusätzlichen Stress. Entwickler haben in der Regel genug damit zu tun, ihre Endtermine einzuhalten.

1.6 Entwicklung einer Anwendungsprogrammschnittstelle

Neben der ganzen Theorie wollen wir eine Anwendungsprogrammschnittstelle (API) von Grund auf entwickeln, um Sie mit den Konventionen und Konzepten vertraut zu machen, die wir zuvor besprochen haben. Dabei verfolgen wir keinen theoretischen, sondern einen praktischen Ansatz. Diese Vorgehensweise haben wir gewählt, damit Sie sich die einzelnen Schritte merken können. In zukünftigen Projekten müssen Sie APIs auf einer theoretischen Grundlage konzipieren, ohne vorher eine einzige Programmzeile gesehen zu haben. Hinweise, Tipps und Tricks finden Sie im Kapitel 3 »Programmentwicklung: ein Praxisbeispiel«.

Das Modul, für welches wir eine API generieren wollen, soll einen einfachen Terminkalender steuern. Wie die Funktionen des Terminkalenders tatsächlich implementiert werden, interessiert an dieser Stelle nicht. Genaugenommen ist dies genau der Teil, den der Benutzer nicht sehen soll. Der Benutzer möchte lediglich eine Reihe von Zuordnungen vornehmen. Die API muss sich also als genau dies darstellen, nämlich als Schnittstelle, über welche Zuordnungen verwaltet werden können. Es ist nicht erforderlich, dem Benutzer das zugrundeliegende System zu erklären – ob Sie den Julianischen, den Gregorianischen Kalender oder ein ganz eigenes Format verwenden. Vielleicht möchten Sie dem Benutzer irgendwann eine Reihe zusätzlicher Funktionen zur Verfügung stellen (z.B. zur Konvertierung des Datenformats), aber dies ist vollkommen überflüssig, wenn Sie den Benutzer zunächst nur in die Lage versetzen wollen, Zuordnungen vorzunehmen.

Andererseits bedeutet dies nicht, dass Sie die zukünftige Einbindung solcher Funktionen verhindern müssen. Die Kunst, eine Schnittstelle zu entwickeln, besteht darin, sie so zu konzipieren, dass sie Ihren Anforderungen genau entspricht und trotzdem später erweitert werden kann. Dies erfordert eine sorgfältige Planung und gut durchdachte Definitionen, worauf wir in diesem Kapitel immer wieder hingewiesen haben.

Für den Benutzer muss die API als einziger Weg erscheinen, die Funktionalität des Moduls zu erreichen, die sie repräsentiert. Keine Funktion fehlt und es gibt keine unnötige oder nicht direkt zum Modul gehörende Funktion.

Folgende Anforderungen können an einen einfachen Terminkalender gestellt sein:

- Ein Ereignis soll hinzugefügt werden.
- Ein Ereignis soll gelöscht werden.
- Eine Liste neuer Ereignisse soll abgefragt werden.

Zunächst wollen wir Prototypen entwickeln, Funktionen hinzufügen oder entfernen. Betrachten Sie dazu Listing 1.10. Welche Informationen benötigen diese Funktionen möglicherweise und welche Werte geben Sie wohl zurück?

```
void add_an_event(int day, int month, int year, int hour, int minutes,
➥int seconds, string description);
void delete_an_event(int day, int month, int year, int hour, int minutes,
➥int seconds);
```

Listing 1.10: Prototypen für die ersten beiden Funktionen

Wahrscheinlich kommt Ihnen spontan Folgendes in den Sinn: Dies ist eine Schnittstelle, die eine »gewöhnliche« Liste von Parametern akzeptiert, nämlich das Datum als Tag / Monat / Jahr-Variablen und die Zeit in Form von Stun-

den/Minuten/Sekunden sowie eine Zeichenkette für die Beschreibung der Zuordnung. Zurückgegeben wird bei den Funktionen nichts. Die Namen sind sprechend.

Sprechend, ja – aber auch klar? `add_an_event()` ist sicherlich aussagekräftig, aber dennoch nicht sehr glücklich gewählt für eine solche Funktion. Vor allem, weil die Funktion für den globalen Zugriff gedacht ist. Dies bedeutet, es ist das Hauptelement für die API. Dementsprechend sollte es durch einen Präfix als zur API zugehörig gekennzeichnet sein.

Wie könnte dieses Präfix heißen? `calendar` und `scheduler` sind gute Alternativen; in diesem Beispiel verwenden wir `calendar` (siehe Listing 1.11).

```
void calendar_add_an_event(int day, int month, int year, int hour, int
➥minutes,int seconds, string description);
void calendar_delete_an_event(int day, int month, int year, int hour, int
➥minutes,int seconds);
```

Listing 1.11: Umbenannte Funktionsprototypen

Nun haben wir ein Präfix, aber die eigentlichen Namen sind noch unbefriedigend. Das `an` in `calendar_add_an_event()` und genauso in `calendar_delete_an_event()` ist eigentlich überflüssig; es ist ein Relikt aus der Zeit, als man »zu sprechende Namen« wählte. Eine gute Taktik ist, ganz allgemein, bei der Auswahl von Funktionsnamen, Wörter wie »an« bzw. »ein« wegzulassen. Meistens nehmen diese Wörter nur Platz weg, machen aber keinen großen Unterschied, weil sie keine erläuternde Funktion haben. Bei der Definition von Variablennamen sollten Sie solche Wörter tunlichst vermeiden. Es macht überhaupt keinen Sinn, eine Variable `$a_key` oder `$the_key` zu nennen, denn es ist offensichtlich, dass es sich um einen Schlüssel handelt. Sinnvoller ist es zu beschreiben, welche Art von Schlüssel es ist, z.B. `$last_user_key`.

Listing 1.12 zeigt die nochmals umbenannten Funktionen

```
void calendar_add_event(int day, int month, int year, int hour, int
➥minutes,int seconds, string description);
void calendar_delete_event(int day, int month, int year, int hour, int
➥minutes,int seconds);
```

Listing 1.12: Endgültige Funktionsnamen

Auf zum nächsten Punkt. Diese Funktionen haben eine lange Parameterlisten. Ist dies notwendig? Die Parameter wurden, so wie sie oben zu sehen sind, intuitiv ausgewählt, und zwar nach dem herkömmlichen Datenformat, das in Tag, Monat, Jahr, Stunde, Minute und Sekunde unterteilt ist. Ein Datenaustausch über eine solche Schnittstelle ist jedoch relativ komplex. Eine Funktion sollte normalerweise nicht mehr als fünf Parameter akzeptieren. Wenn mehr Parameter übergeben werden müssen, sollten Sie sich überlegen, ob Sie diese

nicht strukturiert übergeben. Mit Strukturen halten Sie die Schnittstelle »sauber«. Dies ist manchmal wichtiger als die Bemühung, die zusätzliche Belastung zu vermeiden, die durch Strukturen entstehen, wenn sie initialisiert und / oder modifiziert werden.

Bevor Sie versuchen, alle Parameter in einer Struktur zusammenzufassen, sollten Sie überlegen, ob Sie ein anderes Datenformat verwenden können. Zur Kodierung von Datum und Zeit können Sie beispielsweise den binären Dezimalcode (BCD, Binary Coded Digits) oder das Timestamp-Format für UNIX verwenden. In beiden Fällen werden die erforderlichen Variablen in eine einzige gepackt. BCD ist zum Teil immer noch weit verbreitet, aber im Falle von PHP, das ursprünglich auf einer UNIX-ähnlichen Plattform entwickelt wurde, dominiert ganz klar die Timestamp-Methode (siehe Listing 1.13). Sollte Ihnen dieses Verfahren nicht geläufig sein: Timestamps zählen die Sekunden, die seit Mitternacht des 1. Januar 1970 UTC vergangen sind. Sie werden als Dezimalzahl in 32-Bit-Schreibweise ausgedrückt. Dies führt zu einem Überlauf im Jahr 2106. Da PHP nicht auf die 32-Bit-Schreibweise für die Erfassung der Timestamps festgelegt ist, könnte es die Größe der Timestamps auf 64-Bit ändern und wäre damit Jahr 2106-fest. Ihr Programm wird es nicht bemerken.

Ein weiterer Vorteil von Timestamps besteht darin, dass ein Großteil der PHP-Funktionen diese wieder in für den Menschen lesbare Daten zurückverwandelt. Damit können Sie mit Timestamps leichter Berechnungen anstellen. Um beispielsweise die Zeitdifferenz zwischen zwei Ereignissen festzustellen, brauchen Sie nur ein Timestamp vom anderen abzuziehen.

```
void calendar_add_event(int timestamp, string description);
void calendar_delete_event(int timestamp, int seconds);
```

Listing 1.13: Korrigierte API

Wie Sie sehen, kann es sehr wichtig sein, Formate und Methoden zu überprüfen, die zur Bearbeitung spezieller Daten zur Verfügung stehen. Das aktuelle Format kürzt die Argumentliste nicht nur auf 1/7, sondern ist – zufälligerweise – auch das maschinenspezifische Format der zugrundeliegenden Architektur für die Bearbeitung des Datums und der Zeit. Die Überprüfung maschinenspezifischer Formate und bestehender Standards ist ein Schritt, den Sie in Ihrer Vorbereitungszeit nie unterschätzen sollten. In der Entwicklungsphase sollte Ihnen Nichts einfach »zufällig« passieren. Kenntnis der Materie ist Pflicht.

Behalten Sie dies im Hinterkopf, wenn Sie sich jetzt die dritte, laut Liste benötigte Funktion ansehen. Sie dient dazu, eine Liste der neuen Ereignisse abzufragen. Hier bekommen wir Probleme, weil die Ausgabe kein einzelner Wert, sondern eine variable Liste von zugeordneten Werten ist:

Timestamp 1 =>Beschreibung 1

Timestamp 2 =>Beschreibung 2

Timestamp 3 =>Beschreibung 3

Die Rückgabe der Daten könnte durch Referenz auf die Parameter erfolgen. Näheres zu diesem Thema erfahren Sie im Kapitel 2 »Erweiterte Syntax«.

```
//
// List function in pseudocode
//

function calendar_get_event_list($range, &$timestamp, &$description)
{

    while($current_timestamp < $range)
    {
        $timestamp[] = $next_event_timestamp;
        $description[] = $next_event_description;
    }

}
```

Angesichts der großen Bandbreite neuer Ereignisse würde dieser Pseudocode zwei Arrays ausfüllen, nämlich `$timestamp` und `$description`. Demnach würde der Timestamp für Ereignis 1 in `$timestamp[0]` und die Beschreibung von Ereignis 1 in `$description[0]` gespeichert.

Dies ist jedoch eine suboptimale Lösung, denn es zeugt von schlechtem Stil, zusammenhängende Elemente in zwei separaten Variablen zu verwalten. Besser ist, zusammenhängende Elemente durch einen Gruppendatentyp zu erfassen, entweder in einer Klasse (der einzigen Möglichkeit, in PHP strukturierte Typen zu generieren) oder in einem assoziativen Array.

Assoziative Arrays haben den Vorteil, dass sie sowohl per Schlüssel (dem indizierenden Element – in regulären Arrays 0, 1, 2, 3 usw.) als auch per Wert (der informativen Komponente) durchsucht werden können, aber keine vordefinierte Struktur haben. Vielmehr haben sie eine variable Struktur, die quasi im Vorübergehen geändert werden kann. Dies kann allerdings zu Daten führen, die nicht unbedingt eine gültige Struktur haben und umständlich zu verwalten sind.

Klassen haben den Vorteil, dass ihre Struktur absolut sichtbar ist. Allerdings benötigen Sie einen vordefinierten Datentyp. Wenn wir nun einen Datentyp für den zurückzugebenden Wert definieren, sollten wir diesen Datentyp aus Konsistenzgründen auch dazu verwenden, um Ereignisse zu generieren und zu löschen. Dazu müssten wir allerdings die bestehenden Funktionen im Nachhinein ändern – es ist wenig sinnvoll, eine einzige Funktion hinzuzufü-

gen. Wie Sie sehen, hätten wir durch eine sorgfältige Planung viel Zeit sparen können. Wenn wir zunächst einen Strukturtyp für Zuordnungen definiert hätten und danach die ersten beiden Funktionen, hätten wir diesen Datentyp dafür verwenden können. Dadurch hätten wir eine exakte Lösung erhalten, die wir nun in unsere Listenfunktion implementieren könnten.

Da eine Klasse ein Stilbruch im Programmcode bedeuten würde, verwenden wir das assoziative Array. Die Listenfunktion wird keinen Fehlercode zurückgeben, also verwenden wir den zurückgegebenen Wert der Funktion, um die Daten zum Anrufer zurückzuschicken. Wenn Sie Fehlercodes verwenden möchten, sorgen Sie dafür, dass alle Funktionen Fehlercodes zurückgeben, auch wenn die Übertragung immer klappt. Der Benutzer Ihrer API weiß normalerweise nicht, welche Funktion fehlschlagen kann, und erwartet, dass alle Funktionen einen Fehlercode zurückgeben, wenn einige dies bereits getan haben. Arbeiten Sie ein konsistentes Fehlercode-Schema aus – mehr dazu finden Sie in Kapitel 3.

Zurück zur Listenfunktion. Der zukünftige Prototyp könnte folgendermaßen aufgebaut sein:

```
function calendar_get_event_list($range)
{
    // Retrieve event list
}

$event_list = calendar_get_event_list($required_range);

for($i = 0; $i < count($event_list); $i++)
    print("Event at $event_list[$i]["time"]: $event_list[$i]["text"]<br>");
```

Die Ausgabe hierzu könnte folgendermaßen aussehen:

```
Event at 95859383: Team meeting
Event at 95867495: Deadline for Telco project
Event at 95888371: XML Seminar
```

Sieht gut aus, aber dieser Programmteil hat noch einen schwerwiegenden Fehler. In der `for()`-Schleife werden die Daten in einem zweidimensionalen Array zurückgegeben, wobei die assoziativen Schlüssel `time` und `text` verwendet werden. Die Variablen wurden zuvor aber anders benannt, und zwar `$timestamp` für die Zeit und `$description` für den beschreibenden Text. Verwenden Sie zur Speicherung eines assoziativen Arrays für die Schlüssel stets dieselben Namen, die Sie für die entsprechenden Variablen gewählt haben. Im obigen Fall sollte die `for()`-Schleife also folgendermaßen auf das Array zugreifen:

```
function calendar_get_event_list($range)
{
    // Retrieve event list
```

```
}

$event_list = calendar_get_even_list($required_range);

for($i = 0; $i < count($event_list); $i++)
    print("Event at $event_list[$i]["timestamp"]:
    ➥$event_list[$i]["description"]<br>");
```

1.7 Zusammenfassung

Die Entwicklung eines Programms beinhaltet sehr viel mehr, als nur den reinen Code herunterzuschreiben, die richtige Syntax einzuhalten und sicherzustellen, dass das Programm läuft. Da das Programm nicht nur vom Computer, sondern auch von anderen Programmierern (wie Ihnen) gelesen wird, sollte der Quellcode klar und präzise sein. Ein gut geschriebenes Programm ist einfach zu lesen, ausführlich kommentiert und verwendet keine exotischen Ausdrücke. APIs sollten eine klare und konsistente Schnittstelle zur Verfügung stellen, in logische Einheiten unterteilt sein und den Backend abstrahieren. Da größere Projekte selbst mit dem klarsten Programmcode nicht selbsterklärend sind, sollte hierfür unbedingt eine technische Dokumentation erstellt werden.

Die in diesem Kapitel vorgestellten Konventionen zur Kodierung basieren nicht auf bindenden Regeln, sondern entstammen dem Erfahrungsschatz vieler routinierter Programmierer. Sie sind nicht schwer zu befolgen – und sie machen Ihnen und Ihren Kollegen das Leben ein gutes Stück leichter.

Nähere Dich ihm und es gibt keinen Anfang;
folge ihm und es gibt kein Ende.
Du kannst es nicht wissen,
aber sein,
frei in Deinem eigenen Leben.

Pearson
Education

ADDISON-WESLEY

2 Erweiterte Syntax

Die Welt wurde aus der Leere gebildet,
wie Utensilien aus einem Holzblock.
Der Herr kennt die Utensilien,
hält sich jedoch an den Block:
damit kann er alle Dinge benutzen.

Wie in Kapitel 1 erwähnt, glauben wir, dass wir zur wirklichen Beherrschung einer Programmiersprache nicht nur deren Syntax und Semantik verstehen müssen, sondern auch ihre Philosophie, den Hintergrund und die Hauptmerkmale der Struktur. Um PHP zu meistern, müssen Sie auch alle seine Eigenheiten gut kennen.

2.1 PHP-Syntax

PHP ist eine Mischung aus verschiedenen Sprachen. Sie können einen starken Einfluss von C erkennen (einige sagen Java, aber Java ist auch aus C entstanden). Während die Syntax von PHP stark durch C beeinflusst ist, unterscheidet sich die PHP-Semantik von der von C sehr stark. PHP wird interpretiert und hat keine strengen Variablentypen. Wenn Sie auf eine Variable referenzieren, wird sein Typ im »Vorübergehen« ermittelt und so behandelt, wie es die gegenwärtige Situation erfordert. Dies ist zwar eine etwas vereinfachte Erklärung, Sie sollten sie aber während der Entwicklung im Hinterkopf behalten.

PHP ist eine interpretierte Sprache, in welcher der Programmcode ausgewertet und Schritt für Schritt ausgeführt wird. Zusammen mit der Art und Weise, wie PHP Variablen verwaltet, erhalten Sie damit vielfältige Möglichkeiten für die Programmierung, jedoch auch eine Menge Fallen. Dieses Kapitel befasst sich mit den typischen Dingen, die man tun bzw. lassen sollte, um erweiterte syntaktische und Algorithmenfunktionen zu erhalten:

- Definition von Konstanten
- Array-Funktionen
- Klassen
- Verknüpfte Listen
- Bäume
- Assoziative Arrays
- Mehrdimensionale Arrays
- Variable Argumente und variable Argumentenlisten

- Variable Variablennamen
- Variable Funktionsnamen
- Selbstmodifizierender Highlevel-Code
- Polymorphisierung

Auf all diese Themen werden wir in den folgenden Abschnitten eingehen.

2.2 Definition von Konstanten

In PHP gibt es zwar keinen Ausdruck und keine Konstruktion, um Konstanten in Form von nicht-modifizierbaren Variablen zu definieren, trotzdem können Sie dieses Ziel durch Verwendung von definierten Werten erreichen. Definierte Werte sollten verwendet werden, um alle festen Werte zu ersetzen, wie z.B. Fehlercodes, Dateiformatkonstanten, spezielle Zeichenfolgen und andere Konstrukte, die eine spezielle Bedeutung für das Programm oder die Bibliothek haben und während der Programmausführung nicht geändert werden sollen.

Definierte Werte haben den großen Vorteil, dass sie die Bedeutung spezieller Werte verdeutlichen und gleichzeitig eine weitere Abstraktionsebene bieten:

```
// read file type from input
$file_type = fgets($file, 32);

// decide what kind of file it is
switch($file_type)
{
    case FT_GIF_IMAGE:   /* handle GIFs here */
                         break;

    case FT_PNG_IMAGE:   /* handle PNGs here */
                         break;

    case FT_ZIP_ARCHIVE: /* handle ZIPs here */
                         break;
}
```

Hinweis: Mit dem Begriff spezielle Werte meinen wir »magische Nummern« oder spezielle Zeichenfolgen. Wenn Sie beispielsweise aus Ihrem Programm auf eine GIF-Datei zugreifen wollen und Ihr Programm intern GIF durch die »magische Nummer« 6 erkennt (weil dies der Programmierer so festgelegt hatte – es könnte auch 1234 sein), können Sie mit `define("GIF_FILE", 6)` für diesen Wert eine Definition erzeugen und künftig auf Ihre »magische Nummer« mit dem Schlüsselwort `GIF_FILE` zugreifen.

Dieser Programmabschnitt liest einen Bezeichner aus einer Eingabedatei und entscheidet dann, wie er auf diesen Bezeichner reagiert. Der Bezeichner gibt an, ob es sich bei den folgenden Daten um ein GIF-Bild, ein PNG-Bild oder ein ZIP-Archiv handelt. Er könnte folgendermaßen aussehen:

GIF-Bild	»GIF_IMG«
PNG-Bild	»PNG_IMG«
ZIP-Archiv	»ZIP_ARC«

Diese Bezeichner würden in einer include-Datei folgendermaßen definiert werden:

```
define("FT_GIF_IMAGE", "GIF_IMG");
define("FT_PNG_IMAGE", "PNG_IMG");
define("FT_ZIP_ARCHIVE", "ZIP_ARC");
```

Diese Einrichtung hat den Vorteil, dass Sie alle Bezeichner an einer zentralen Stelle sammeln können. Wenn Sie einen Bezeichner ändern müssen, brauchen Sie nur seine Definition ändern. Andernfalls müssten Sie den gesamten Programmcode durchsuchen und jede dieser Zeichenfolgen manuell ersetzen. Mithilfe definierter Werte lässt sich diese Arbeit auf eine einzelne Programmzeile reduzieren.

Die Namen der definierten Werte sollten stets in Großbuchstaben geschrieben werden, um sie als solche zu kennzeichnen. In einigen Fällen, wie bei Bibliotheksfunktionen, sollte ihnen außerdem ein Präfix vorangestellt sein. Im vorangegangenen Beispiel haben die Bezeichner das Präfix »FT_« für »file-type« (Dateityp).

Versuchen Sie, definierte Werte so häufig wie möglich einzusetzen. Wenn Sie auf eine Situation treffen, in der Sie normalerweise einen Wert in Ihr Programm fest eingeben würden, sollten Sie bedenken, dass die Eingabe von fixen Werten unvorteilhaft ist. Betriebssysteme oder ähnliche maschinencodeorientierte Programme haben häufig die längsten Definitionslisten, denn um sie portabel zu gestalten, muss jede Kleinigkeit abstrahiert werden. Sie können keine Annahmen über die Byte-Größe, die Wortlänge oder die Registergröße machen; im Prinzip muss alles, was Sie sehen, auf irgendeine Weise abstrahiert werden. Da PHP per se portierbar ist, bedeutet dies, dass es an keine bestimmte Hardware oder andere Umgebungskonfigurationen gebunden ist (sein Interpreter ändert die Umgebung nicht, wie auch immer das zugrunde liegende Betriebssystem aussehen mag). Daher brauchen Sie mit Definitionen in PHP nicht übertreiben. Aber ein gutes Maß zeugt von guten Programmierstil.

2.3 Array-Funktionen

Die wichtigsten Array-Funktionen sind `list()`, `each()` und `count()`.

`list()` ist eine Art Operator, der aus einer Reihe von Variablen einen l-Wert bildet (einen Wert, der auf der linken Seite eines Ausdrucks verwendet werden kann), der sich ähnlich wie ein Element eines mehrdimensionalen Arrays wiederum als neue Entität darstellt. Als Argument kann er eine Liste von Variablen besitzen. Wird ihm etwas zugeordnet (eine Liste von Variablen oder ein Array-Element), dann wird die Liste der Variablen, die als Argument des Operators `list()` angegeben wurde, von links nach rechts abgearbeitet. Diesen Argumenten wird dann ein entsprechender Wert aus dem r-Wert (einem Wert, der auf der rechten Seite eines Ausdrucks verwendet werden kann) zugewiesen. Dies lässt sich am besten anhand eines Beispiels erklären:

```
$result = mysql_db_query($mysql_handle, $mysql_db,
                         "SELECT car_type, car_color, car_speed
                          FROM cars WHERE car_id=$car_id");

list($car_type, $car_color, $car_speed) = mysql_fetch_row($result);
```

Hinweis: Dieser Programmcode wird hier nur als Beispiel verwendet. Es empfiehlt sich nicht, diesen Programmcode in realistische Programme zu integrieren, da er darauf aufbaut, dass die Felder in derselben Reihenfolge bleiben. Wenn Sie die Reihenfolge der Felder ändern, müssen Sie auch die Reihenfolge der Variablen in der Anweisung `list()` ändern. Die Verwendung von assoziativen Arrays und die manuelle Extraktion von Werten bedeutet einen zusätzlichen Programmierungsaufwand, führt aber zu einem stabileren Programmcode. Programmcode, wie der oben gezeigte, sollte nur für Optimierungszwecke implementiert werden.

Die SQL-Abfrage wählt die Werte `car_type`, `car_color` und `car_speed` aus einer Tabelle mit Fahrzeugdaten aus. Das Ergebnis der Abfrage wird anschließend mithilfe von `mysql_fetch_row()` ausgelesen, welches die drei Werte in einem Array zurückgibt. `car_type` wird dabei an die Indexposition 0 geschrieben, `car_color` an Indexposition 1 und `car_speed` an Indexposition 2. Diese Werte werden von links nach rechts gelesen und anschließend Wert für Wert den in der `list()`-Anweisung angegebenen Argumenten zugeordnet.

Damit erhalten Sie die folgenden Zuordnungen:

Listenargument	SQL-Feld
`$car_type`	`car_type` (Array-Position 0)
`$car_color`	`car_color` (Array-Position 1)
`$car_speed`	`car_speed` (Array-Position 2)

Die Anweisung `list()` ist besonders nützlich, wenn Sie eine Sammlung von Werten in einzelne Variablen unterteilen möchten. Dies passiert recht häufig bei der Datenbankprogrammierung. Beachten Sie jedoch, dass `list()` nur als l-Wert und nicht als r-Wert fungieren kann. Sie können `list()` nicht verwenden, um eine Reihe von Variablen auszutauschen. Die folgende Anweisung würde beispielsweise nicht funktionieren:

```
list($var1, $var2) = list($var2, $var1);
```

Die Anweisung `each()` wird häufig mit der Anweisung `list()` verwendet. `each()` arbeitet ein Array ab und gibt die einzelnen Elemente in einer Schlüssel / Wert-Kombination zurück. Dies geschieht durch »Abwandern« des Eingabe-Arrays. PHP weist jedem Array einen internen Zeiger zu. Dieser Zeiger weist zu Beginn auf das erste Element des Arrays. Bei jedem Aufruf von `each()` wird das Element zurückgegeben, auf welches der interne Zeiger weist. Anschließend wird der Zeiger um eine vorgegebene Anzahl von Elementen hochgesetzt.

Das Rückgabeformat des Schlüssel / Wert-Paares ist ein vier Elemente umfassendes Array mit den Schlüsseln »0«, »1«, »key« und »value«. Dies bedeutet, dass es sowohl als indiziertes Array als auch als assoziatives Array verwendet werden kann. Der indexierte Teil des Arrays (mit den Schlüsseln »0« und »1«) enthält den Schlüssel des Quellelements an Indexposition 0; der Wert befindet sich an Indexposition 1. Dieselbe Information erhalten Sie auch über den assoziativen Teil des Arrays. (Genaugenommen ist eine Trennung von assoziativen und indizierten Teil des Arrays hier nicht korrekt, da indizierte Arrays nur eine Sonderform der assoziativen Arrays sind. Theoretisch sind es zwar ganz verschiedene Dinge, aber in der Praxis sind sie in PHP identisch. Weitere Einzelheiten finden Sie weiter unten.) Der Schlüssel des Quellelements ist in »key« angegeben, während der Wert in »value«. steht. Ein Beispiel:

```
$my_array = array("Element 1", "Element 2", "Element 3");
```

Hierdurch wird ein Array mit dem folgenden Inhalt erzeugt:

```
Element 1
Element 2
Element 3
```

Um das Prinzip von `each()` besser zu verstehen, ist es sinnvoll, eine detaillierte Auflistung dieses Arrays zu erstellen:

Schlüssel	Wert
0	Element 1
1	Element 2
2	Element 3

Hier sehen die Auflistung aller Schlüssel/Wert-Paare, die in dem Array $my_array enthalten sind:

```
list($key, $value) = each($my_array);
```

Jetzt wenden wir hierauf each() an. Beim ersten Aufruf wird das erste vier Elemente umfassende Array zurückgegeben, welches das erste Schlüssel/Wert-Paar aus $my_array. enthält. Da wir nur zwei Argumente dem Operators list() angegeben haben, können auch nur die Werte aus dem vier-Elemente-Array zugeordnet werden. Es handelt sich um »0«, den ersten Schlüssel, und »Element 1«, das erste Wertelement aus $my_array.

Der folgenden Programmcode listet den Inhalt eines Arrays auf:

```
$my_array = array("Element 1", "Element 2", "Element 3");

while(list($key, $value) = each($my_array))
    print("Key: $key, Value: $value<br>");
```

Das Skript erzeugt eine Ausgabe, wie jene, die in Abbildung 2.1. zu sehen ist.

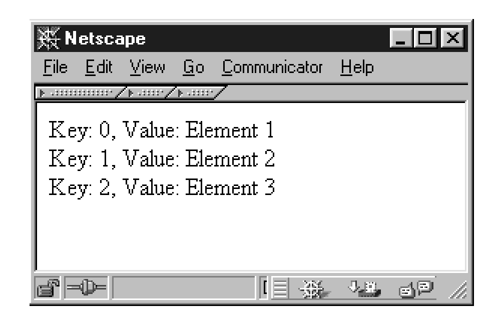

Abbildung 2.1: Array-Auflistung mit each( )

Sie können each() auch verwenden, um die Elemente anzuzeigen, die each() selbst zurückgibt:

```
$my_array = array("Element 1", "Element 2", "Element 3");

while($four_element_array = each($my_array))
{
    while(list($key, $value) = each($four_element_array))
        print("Key $key, Value $value<br>");
}
```

Damit erhalten Sie das in Abbildung 2.2 dargestellte Ergebnis.

Sie können deutlich sehen, dass die Einträge »1«, »value«, »0« und »key« in der Ausgabe, sowie »0« und »1« zusammen nur als »key« und »value« verwendet werden sollten. Beachten Sie, dass jedes Paar einen Eintrag des Quell-Arrays darstellt.

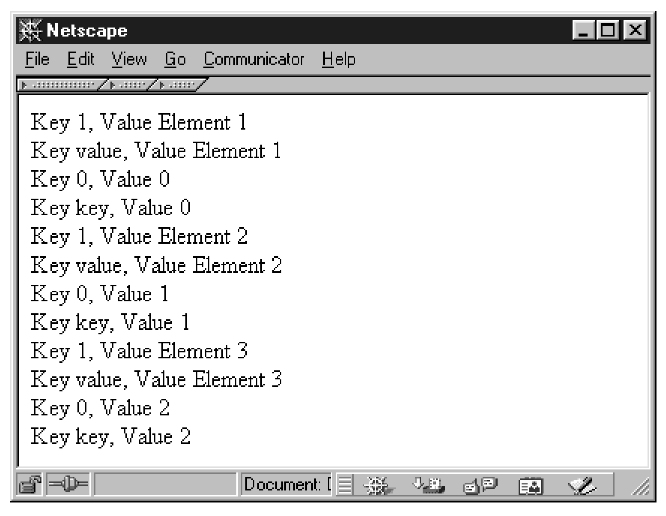

Abbildung 2.2: Rückgabewerte von `each()`

`each()` für ein indexiertes Array zu verwenden scheint zunächst keinen Sinn zu machen, da die Elemente eines indexierten Arrays sehr viel leichter mithilfe einer `for()`-Anweisung gelesen werden können. Dabei gibt es jedoch einige Tücken. Zunächst ist in PHP das indexierte Array nur eine Sonderform des assoziativen Arrays. PHP lässt auch nicht-zusammenhängende Array-Indizes zu, d.h. Sie könnten auch folgendes Array haben:

Schlüssel/Index	Wert
0	Landon
3	Graeme
4	Tobias
10	Till

Bei diesem Array sind nur die Indizes 0, 3, 4 und 10 in Gebrauch, der Rest ist einfach nicht zugeordnet. Mithilfe der Funktion `count()` (welche die Anzahl der zugeordneten Elemente in einem Array zurückgibt) erhalten Sie zwar vier korrekt zugewiesene Elemente, aber Sie können keine `for()`-Anweisung für dieses Array verwenden, da Sie die entsprechenden Schlüssel für die Werte nicht kennen:

```
$my_array = array(0 => "Landon", 3 => "Graeme", 4 => "Tobias", 10 =>
➥"Till");

for($i = 0; $i < count($my_array); $i++)
    print("Element $i: $my_array[$i]<br>");
```

Damit erhalten Sie die in Abbildung 2.3 gezeigte Ausgabe.

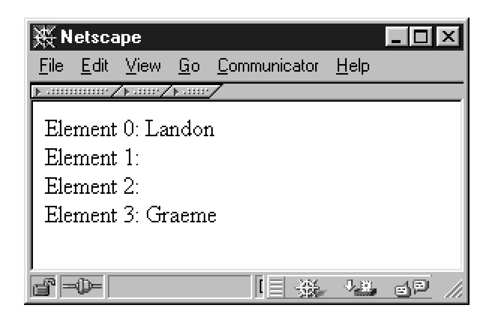

Abbildung 2.3: Ungültiger Zugriff auf ein nicht zusammenhängendes Array

Fehlerberichterstattung

Wenn Sie PHP so eingerichtet haben, dass es ungültige Array-Indizes anzeigt, werden auch ein paar Warnungen ausgegeben. Es empfiehlt sich, während der Entwicklung eine möglichst hohe Ebene für die Fehlerberichte zu wählen.

Eine einfache `for()`-Schleife reicht für Arrays dieser Art nicht aus, denn diese Schleife greift auf Indizes zu, denen noch kein Wert zugewiesen ist. Wenn PHP eine enger definierte Umgebung bereitstellt, würde dies zum Programmabbruch und sofortigen Beendigung des Skripts führen. Wenn Sie den Inhalt und die Konsistenz Ihrer Arrays also nicht genau kennen, sollten Sie `each()` tunlichst nicht verwenden.

`each()` ist auch ein gutes Werkzeug, um sicherzustellen, dass Sie auf Arrays nicht außerhalb ihrer Bereichsgrenzen zugreifen. Auch dies ist ein Grund für Programmunterbrechungen. PHP handhabt Zugriffe außerhalb der Bereichsgrenze auf recht lockere Art (meistens wird nur eine Warnung ausgegeben); trotzdem haben wir es wiederholt geschafft, PHP durch ungültige Array-Zugriffe zum Absturz zu bringen. Im günstigsten Fall reagierte PHP lediglich mit Abbruch des Programms; im schlimmsten Fall belegte das PHP-Modul plötzlich 100% der CPU-Zeit, und der Serverprozess musste abgebrochen werden – eine Situation, die Sie in einer Produktionsumgebung unbedingt vermeiden sollten. In unserem Falle wurde dies wahrscheinlich durch eine fehlerhafte interne Routine zur Array-Bearbeitung in PHP verursacht. Sie sollten trotzdem keine ungültigen Array-Zugriffe provozieren. Es zeugt von schlechtem Programmierstil, und PHP bietet `each()`, `list()` und verwandte Funktionen, um Ihren Programmcode zu sichern. Dies sollten Sie nutzen.

Ursprünglich wurde `each()` zur Benutzung mit »realen« assoziativen Arrays entwickelt, die zur Indexierung von Daten nicht-numerische Schlüssel verwenden. In diesem Fall ist es nicht möglich, auf die gespeicherten Daten ohne eine Funktion zuzugreifen, die alle verfügbaren Schlüssel auflisten kann (sofern Sie nicht wissen, welche Schlüssel in Gebrauch sind). Diese Arrays

könnten auf ähnliche Weise wie das zuvor beschriebene Array organisiert werden, wobei die Schlüssel und Werte in umgekehrter Reihenfolge aufgeführt werden:

```
$my_array = array("Landon" => 1, "Graeme" => 2, "Tobias" => 3, "Till" =>
➥4);

while(list($key, $value) = each($my_array))
    print("Key $key, Value $value<br>");
```

Hier wird das Array nicht über die Nummer, sondern über den Namen indiziert. Wie finden Sie nun heraus, welche Namen in dem Array enthalten sind, wenn Sie keine vordefinierten Schlüssel verwenden? Hier hilft Ihnen `each()`, wie Abbildung 2.4 zeigt.

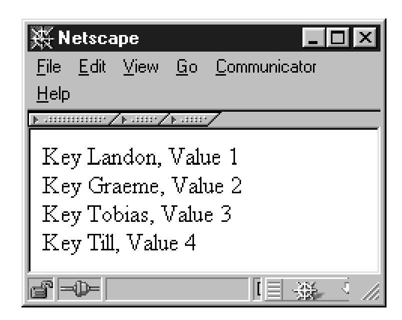

Abbildung 2.4: Assoziative Array-Auflistung mit each( )

Das Ergebnis ist nicht sehr überraschend, trotzdem sehr nützlich zu wissen.

Ein letzter wichtiger Hinweis zu `each()`: Damit Sie ein Schlüssel / Wert-Paar durch Iteration ermitteln können, muss PHP wissen, auf welches Paar Sie zuletzt zugegriffen haben. Wenn Sie also eine weitere Iteration mit demselben Array durchführen, werden in beiden Fällen nicht dieselben Schlüssel / Wert-Paare zurückgegeben. Um den internen Array-Zähler zurückzusetzen, müssen Sie die Funktion `reset()` verwenden.

Diese Funktion setzt den internen Zeiger von PHP wieder auf das erste Element des Arrays, entsprechend wird dessen Wert als Ausgabewert von `reset()` zurückgegeben.

Das folgende Skript greift in zwei verschiedenen Schleifen zweimal auf dasselbe Array zu, wobei beide Male `each()` verwendet wird:

```
$my_array = array("Landon" => 1, "Graeme" => 2, "Tobias" => 3, "Till" =>
➥4);
print("<h2>Looping without reset()</h2>");
print("<h3>First loop</h3>");
for($i = 0; $i < 2; $i++)
    list($key, $value) = each($my_array);
```

```
    print("Key $key, Value $value<br>");
}
print("<h3>Second loop</h3>");
for($i = 0; $i < 2; $i++)
    list($key, $value) = each($my_array);
    print("Key $key, Value $value<br>");
}
```

Wie die Ausgabe in Abbildung 2.5 zeigt, beginnt die zweite Schleife nicht wieder beim ersten Element. Statt dessen setzt sie dort an, wo die erste Schleife beendet wurde. Dies liegt daran, dass der interne Array-Zeiger von PHP nicht zurückgesetzt wurde. Eine leichte Veränderung des Skripts verändert das Ergebnis (siehe Abbildung 2.6):

```
$my_array = array("Landon" => 1, "Graeme" => 2, "Tobias" => 3, "Till" =>
➥4);
print("<h2>Looping with reset()</h2>");
print("<h3>First loop</h3>");
for($i = 0; $i < 2; $i++)
{
    list($key, $value) = each($my_array);
    print("Key $key, Value $value<br>");
}
print("<h3>Calling reset()</h3>");
reset($my_array);
print("<h3>Second loop</h3>");
for($i = 0; $i < 2; $i++)
{
    list($key, $value) = each($my_array);
    print("Key $key, Value $value<br>");
}
```

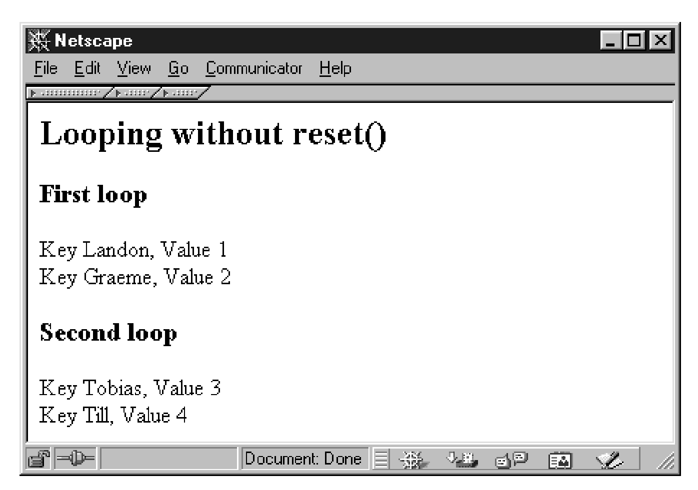

Abbildung 2.5: Verwendung von each() in Verbindung mit reset().

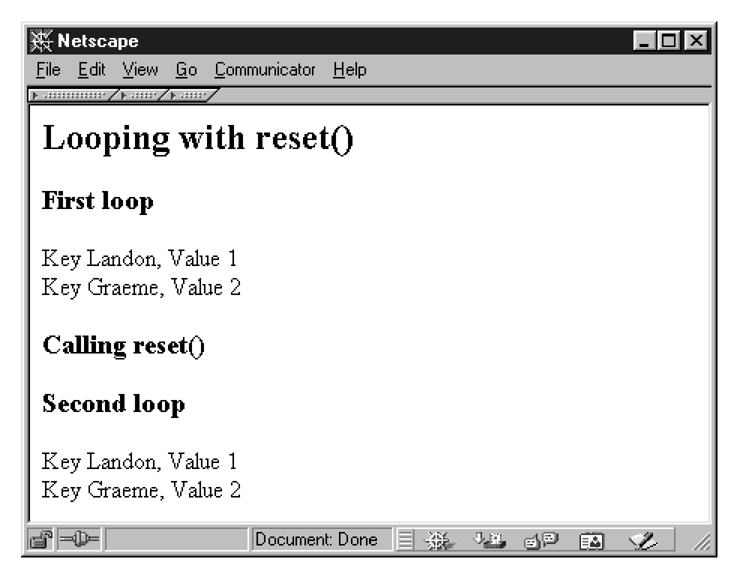

Abbildung 2.6: Verwendung von each() in Verbindung mit reset().

Zwischen den beiden Schleifen wurde ein Aufruf von each() eingefügt. Die zweite Verwendung von each() beginnt beim Lesen des Schlüssel / Wert-Paares aus dem ersten Element.

Neben reset() gibt es noch eine Reihe weiterer Funktionen, um die Aktivität von each() aufzuteilen: next(), prev() und current(). Mithilfe dieser Funktionen können Sie ein Array manuell in beide Richtungen abarbeiten. next() gibt das aktuelle Element zurück und setzt den internen Array-Zeiger auf den nächsten Wert; prev() tut das Gleiche, aber setzt den Array-Zeiger auf den vorhergehenden Wert. current() gibt nur das Array-Element zurück, auf welchem der Zeiger gerade steht. Diese Funktionen geben jeweils den Wert false zurück, wenn sie auf ein leeres oder nicht zugeordnetes Element treffen, oder am Ende des Arrays angelangt sind. Es gibt keine Möglichkeit, diese beiden Fälle zu unterscheiden. Daher sollten diese Funktionen nur verwendet werden, wenn each() nicht anwendbar ist und die Möglichkeit, auf ein nichtzugeordnetes Element zu treffen, ausgeschlossen werden kann.

Neben den soeben beschriebenen Funktionen befassen sich noch eine Reihe weiterer PHP-Funktionen mit Arrays. Eine ausführliche Beschreibung finden Sie im Benutzerhandbuch. PHP 4.0 hat eine Vielzahl neuer Array-Funktionen. Auf sie alle einzugehen, würde uns hier zu weit vom eigentlichen Thema abbringen.

2.4 PHP und objektorientierte Programmierung

Anfang der 90er Jahre lernten die meisten bekannten Compiler – wie z.B. die Familie der Borlandcompiler – mit objektorientierten Programmierungserweiterungen (OOP) von »Basissprachen« wie Pascal und C umzugehen. Plötzlich wurden Begriffe wie Klassen, Objekte, Schablonen und Vererbung zu den Schlagworten schlechthin, den heißesten Themen der Softwareentwicklung. OOP war hip und viele Firmen sprangen auf den Zug auf, indem sie ihre Softwarepakete von der prozeduralen auf objektorientierte Programmierung umstellten. Inzwischen ist die Euphorie verblasst, aber eine Sprache, die keine Objekte handhaben kann, wird immer noch als veraltet betrachtet. PHP unterstützt Objekte. Im folgenden Abschnitt wollen wir die Vor- und Nachteile der objektorientierten Programmierung mit PHP erörtern.

Wir halten die Entwicklung zu einer rein objektorientierten Softwareentwicklung für ein wenig zweifelhaft. Große Programmpakete wurden unter erheblichem finanziellem Aufwand in Objekte umgewandelt – nicht eingerechnet der erhebliche Zeitaufwand der Entwickler für die Neukonzeptionierung, Umstrukturierung und Neuinstallation von Tausenden von Programmzeilen. Diese Programmpakete liefen mit prozeduraler Programmierung hervorragend und einige von ihnen benötigten überhaupt keine Objektunterstützung. Wir haben Werbung für Softwaresicherheitssysteme gesehen, die besagte »nun komplett neu implementiert mit objektorientierter Programmierung«. Alles, was diese Systeme taten, war einen Dongle (ein kleines Gerät, das in den Parallelport eines PC eingesteckt wird und als Hardwareschlüssel dient) zu überprüfen, möglicherweise nach einem Passwort zu fragen sowie in einigen Fällen die ausführbaren Programme zu verschlüsseln, mit denen sie verknüpft waren. Die Anwendungen, die sich auf diese Pakete stützten, mussten häufig nicht einmal eine besondere Prozedur starten, um die Passwortüberprüfung einzuleiten. Einige wurden automatisch ausgeführt, sobald die Anwendung gestartet wurde. In anderen Fällen wurde die Überprüfung der Umgebung auf den Aufruf einer einzelnen Funktion beschränkt. Wer braucht jedoch ein Objekt für eine einzelne Funktion? Intern war die Ausführung des Programmcodes streng linear und der Hardwarezugriff wurde von separaten Prozeduren verwaltet (und einige der Anzeigen haben schlichtweg gelogen).

Der fragwürdigste Fall, dem wir begegneten, war ein Entwickler, der an einer Grafikbibliothek arbeitete, um komplexe mathematische zwei- und dreidimensionale Objekte zu zeichnen. Er begründete seine Entscheidung, Objekte zu verwenden, damit, dass er einen Borland-Vertreter auf einer Konferenz getroffen hatte und dieser ihm die objektorientierte Programmierung empfahl. Aber ist es der richtige Weg, die Entwicklung von verlässlichen Softwaresystemen nur auf Empfehlungen zu basieren?

Welche Vorteile haben Objekte nun, und inwiefern unterscheidet sich OOP von dem prozeduralen Ansatz? Warum sollten wir uns überhaupt damit befassen?

Die letzte Frage zuerst. Wir sollten uns gründlich überlegen ob wir in PHP eine objektorientierte Programmierung einsetzen, denn es macht wenig Sinn, eine Technik zu verwenden, die eine größere Belastung für die Entwicklung darstellt, von der zugrunde liegenden Architektur nicht ausreichend unterstützt wird und für die Anwendung im Grunde genommen keinen Unterschied macht. Prozedurbasierte Projekte können genauso effektiv, leicht zu pflegen und erweiterbar sein wie objektorientierte Projekte. Tabelle 2.1 zeigt die wichtigsten Unterschiede zwischen den beiden Ansätzen.

Objekte	Prozeduren
Vollständige Verschachtelung von Daten	Keine Verschachtelung von Daten; nur über separate Parameter
Mehrere Instanzen sind zugelassen	Mehrere Instanzen nicht erlaubt; unterschiedliche Datensätze müssen durch Kopien der vorhandenen Variablen verwaltet werden
Ermöglicht zusätzliche Funktion über Vererbung bei Erhalt der Schnittstelle	Keine Vererbung; zusätzliche Funktionalität nur über die API durch Bereitstellung einer weiteren API-Schicht oder durch Ändern der gesamten API
Selbstzentriert; das Objekt hat seinen eigenen Datensatz und muss nur für dessen Gültigkeit sorgen und den Zugriff für andere Parteien gewährleisten	Global ausgerichtet; Prozeduren haben keine eigenen Datensätze. Die Daten werden vom Anrufer bereitgestellt.
Bietet einfache Mittel zur Sicherstellung der Datenintegrität	Initialisierung und Cleanup (Konstruktor/Destruktor) Datenintegrität kann nur schwer sichergestellt werden. Initialisierung und Cleanup müssen explizit aufgerufen werden.
Isolierter Namensraum	Namen müssen in den globalen Namensraum eingeführt werden

Tab. 2.1: Objektorientierte Programmierung versus prozedurale Programmierung

Diese Tabelle führt nur die wichtigsten Unterschiede auf. Es gibt noch weitere, aber wir können bereits sehen, dass es für Prozeduren nicht besonders gut aussieht. Sind Prozeduren wirklich so schlecht, wie es den Anschein hat? Bedeutet dies, dass die objektorientierte Programmierung die prozedurale Programmierung komplett ablöst? Tatsächlich hängt dies von Ihren Zielen und der Plattform ab, auf der Sie arbeiten. In unserem Fall ist die Plattform

PHP. PHP unterstützt natürlich Objekte, jedoch auf eine sehr spezielle Weise. Dies hängt damit zusammen, wie der Interpreter Variablen handhabt.

Sobald PHP auf eine Anweisung trifft, für die ein Schreibzugriff auf eine Variable benötigt wird, wertet es die Daten aus, die in die Variable geschrieben werden, berechnet sie, kopiert diese und weist das Ergebnis dem Zielspeicherbereich zu. (Diese Beschreibung ist zwar etwas vereinfacht, vermittelt Ihnen aber das Grundprinzip.)

```
$some_var = 2;

$my_var = $some_var * 3;

$new_var = $some_var + $my_var;
```

Laut diesem Skript wird PHP

- einen Speicherbereich für `$some_var` erzeugen und »2« hineinschreiben.
- einen Speicherbereich für `$my_var` erzeugen, den Wert von `$some_var` auslesen, ihn mit 3 multiplizieren und das Ergebnis dem neu angelegten Speicherbereich zuordnen.
- einen Speicherbereich für `$new_var` erzeugen, die Werte von `$some_var` und `$my_var` auslesen, sie zusammenzählen und sie wieder in den neuen Speicherbereich zurückschreiben.

Dies klingt alles sehr schön und logisch – es handelt sich jedoch um einfache Typen, mit denen Sie bereits viele Male gearbeitet haben. Dies sieht ganz anders (und unlogisch) aus, sobald PHP mit Klassen zu tun hat:

```
class my_class
{
    var $var1, $var2, $var3;
}

$my_object = new my_class;

$my_object->var1 = 1;
$my_object->var2 = 2;
$my_object->var3 = 3;

$new_object = $my_object;

$new_object->var1 = 3;
$new_object->var2 = 2;
$new_object->var3 = 1;

print("My object goes $my_object->var1, $my_object->var2,
➥$my_object->var3 !<br>");
```

```
print("New object goes $new_object->var1, $new_object->var2,
➥$new_object->var3 !<br>");
```

Was glauben Sie, wird hier als Ergebnis herauskommen? Das Skript deklariert zunächst eine Klasse, erzeugt eine Instanz davon und weist seinen drei Elementen Werte zu. Anschließend erzeugt es eine neue Referenz auf dieses Objekt, weist seine Elemente neu zu und gibt jedes Element aus, wobei es beide Referenzen verwendet. Bedenken Sie, es handelt sich um *eine* Instanz. Abbildung 2.7 zeigt das Ergebnis.

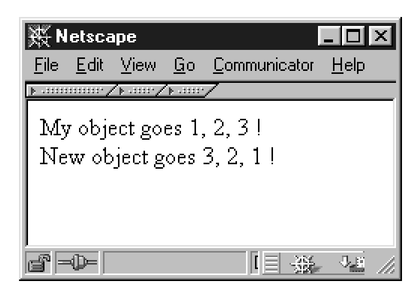

Abbildung 2.7: Erzeugen einer Kopie anstelle einer Referenz mit PHP

Wenn Sie jetzt nicht überrascht sind, kennen Sie PHP bereits recht gut oder haben noch nicht intensiv genug über Objekte nachgedacht. PHP hat nicht einfach eine neue Referenz von `my_class` erzeugt sondern eine Kopie. Dies ist nicht das gewünschte Verhalten, da der Operator `new` eigentlich im Speicherbereich eine Instanz von `my_class` erzeugen und einen Verweis auf diesen zurückgeben sollte. Wenn Sie diese Referenz einer anderen Variablen zuordnen, sollte damit nur die Referenz kopiert werden und die Originaldaten unberührt bleiben, vergleichbar mit einer Dateisystemverknüpfung, die einen Zugriff auf dieselben Daten von verschiedenen Speicherorten im Dateisystem ermöglicht. Dieses Verhalten von PHP – d.h. anstelle der Referenz Kopien referenzierter Daten zu erzeugen – klingt nicht so bedeutend, als dass wir uns darauf konzentrieren sollten. Sie werden jedoch in Kürze feststellen, dass es durchaus von Bedeutung ist.

Hinweis: Zum Zeitpunkt, da dieses Handbuch erstellt wurde, verwendeten sowohl PHP 3.0 als auch PHP 4.0 die Kopiersyntax. Ein Gespräch mit einem Mitglied aus dem Kernentwicklungsteam ergab, dass dieses Standardverhalten geändert werden und künftig eine Referenzsyntax verwendet werden soll. Diese Änderung würde jedoch zum Verlust der Rückwärtskompatibilät führen. Die Änderung ist für die Version 4.1 geplant. Danach wären alle Informationen hier für künftige Versionen nicht mehr gültig.

Warnung: PHP 3.0 verfügt über keinen vernünftigen Mechanismus zur Speicherbereinigung. Sobald Sie einen Wert in eine Variable schreiben, wird nicht der alte Speicherbereich wiederverwendet, sondern neuer Speicher zugewiesen. Mit `unset()` können Sie dies in begrenztem Maße umgehen – es wird zwar kein Speicher freigegeben, sondern nur als wiederverwendbar markiert – auf lange Sicht wird der Speicher Ihres Servers jedoch insbesondere durch langfristige Projekte belegt. Wenn Sie langfristige Skripte verwenden wollen, dann stellen Sie sicher, dass Sie Ihre Datenbankergebnisse beispielsweise mit `mysql_free_result()` freigeben und `unset()` auf alle Variablen anwenden, die keine wertvollen Informationen mehr enthalten. Der Speicher wird jedoch erst frei, wenn das Skript beendet ist!

Nehmen wir beispielsweise eine Baumstruktur. Die Klasse zum Aufbau der Baumknoten sieht folgendermaßen aus:

```
class tree_node
{
    var $left_child, $right_child;
    var $value;
}
```

Dies ist zwar nur ein einfacher Baumknoten, aber er enthält alles, was wir benötigen: eine Verknüpfung zum linken Tochterknoten und eine Verknüpfung zum rechten Tochterknoten sowie eine Variable, die den Inhalt dieses Knotens enthält. Lassen Sie uns nun einen einfachen Baum konstruieren:

```
$root_node = new tree_node;
$left_node = new tree_node;
$right_node = new right_node;

$root_node->value = 1;
$left_node->value = 2;
$right_node->value = 3;

$root_node->left_child = $left_node;
$root_node->right_child = $right_node;
```

Dieser Programmcode erzeugt einen Baum mit einem Stammknoten und zwei Tochterknoten, wobei jedem dieser Knoten ein anderer Wert zugewiesen wird. Die Abarbeitung des Baums könnte mit einer Funktion wie dieser erfolgen:

```
function traverse_tree($start_node)
{

    $node = $start_node;
    print("Value is $node->value<br>");
```

```
    print("Traversing left tree:<br>");
    traverse_tree($node->left);

    print("Traversing right tree:<br>");
    traverse_tree($node->right);

}
```

Lassen wir für einen Moment außer acht, dass diese rekursive Funktion nie enden wird, da unser Baum keine Stopp-Markierung besitzt (die Funktion weiß nicht, welcher Knoten tatsächlich einen Tochterknoten besitzt und welcher nicht). Sehen wir uns seine Funktionsweise an und wie PHP damit umgeht.

Der kritische Punkt ist bereits die erste Zeile, in der `$node` ein Wert zugewiesen wird. Der Parameter `$start_node` gibt die Instanz des Knotens an, von welcher gestartet werden soll, und die Zuordnung zu `$node` erzeugt eine Kopie davon. Für eine Funktion, die den Baum einfach rekursiv abarbeitet und den Knoteninhalt zurückgibt, ist die Tatsache, dass eine Kopie erzeugt wird, nicht von Bedeutung; es kann jedoch wichtig werden, wenn Sie den Baum an einer Stelle ändern möchten.

Nehmen wir nun an, Sie möchten eine Funktion schreiben, die an den Knoten ganz links außen einen weiteren Knoten anhängt. Kein Problem. Schreiben Sie einfach eine rekursive Funktion, welche die Sprünge nach links zählt und den Knoten mit der höchsten Zählerzahl zurückgibt. Ändern Sie dann die Verknüpfung zu den Tochterknoten dieses Objekts und Sie sind fertig. Moment – sind Sie wirklich fertig? Denken Sie einen Moment darüber nach, was Sie geändert haben (siehe Abbildung 2.8). Sie haben die Kopie des Knotens links außen geändert, nicht jedoch den Knoten selbst! Sobald Sie diese Funktion verlassen, gehen Ihre Änderungen unwiderruflich verloren.

$root

<table>
<tr><td colspan="16">Knotendaten</td></tr>
<tr><td colspan="8">Knotendaten des linken Teilknoten</td><td colspan="8">Knotendaten des rechten Unterknoten</td></tr>
<tr><td colspan="4">linke Daten</td><td colspan="4">rechte Daten</td><td colspan="4">linke Daten</td><td colspan="4">rechte Daten</td></tr>
<tr><td colspan="2">L</td><td colspan="2">R</td><td colspan="2">L</td><td colspan="2">R</td><td colspan="2">L</td><td colspan="2">R</td><td colspan="2">L</td><td colspan="2">R</td></tr>
<tr><td>L</td><td>R</td><td>L</td><td>R</td><td>L</td><td>R</td><td>L</td><td>R</td><td>L</td><td>R</td><td>L</td><td>R</td><td>L</td><td>R</td><td>L</td><td>R</td></tr>
<tr><td colspan="16">.
.
.</td></tr>
</table>

Abbildung 2.8: Baumstruktur, erzeugt mithilfe der Kopiersyntax

Sie könnten auch eine Referenz auf diese Instanz zurückgeben und diese ändern, indem Sie den einfachen »Zeiger«-Mechanismus verwenden, den PHP bereitstellt. Dann haben Sie das Objekt selbst geändert und nicht seine

Kopie. Sobald Sie den Baum wieder abarbeiten, werden Sie jedoch etwas Merkwürdiges feststellen: Nichts scheint sich geändert zu haben. Tatsächlich hat sich nichts geändert. Dies liegt daran, dass der Elternknoten des Knotens, den Sie ändern wollten, eine eigene Kopie derselben Daten besitzt! Bedenken Sie, dass `$node->left` nichts Anderes ist als eine weitere Kopie eines Knotens im Baum. Dies ist genau die Kopie, welche die Bearbeitungsroutine auswertet. Die Kopie, die Sie geändert haben, bleibt außerhalb des Baumes und wird schließlich bei einer Speicherbereinigung gelöscht. Um auch diesen Knoten zu ändern, müssen Sie Referenzen an die Elternknoten, die Großelternknoten und die Ur-Großelternknoten weiterleiten. Damit landen Sie bei einer weiteren rekursiven Funktion, die alles Andere als einfach zu programmieren ist und in 99% aller Fälle nicht funktionieren wird.

2.4.1 Klassen: PHP 3.0 versus PHP 4.0

PHP 4.0 hat aus der Unfähigkeit von PHP 3.0 gelernt, Objektreferenzen zu verwalten, und bietet nun »echte« Referenzen. »Echt« ist hier in Anführungszeichen gesetzt, da diese nicht wirklich auf den Speicherbereich zeigen, den die andere Variable belegt. PHP interpretiert diese Variablen nur als Referenzen und verhält sich anders. Eingebettet in das vorangegangene Beispiel, sähe der Programmcode folgendermaßen aus:

```
// create multiple references to the original object
$new_object = &$my_object;
$another_object = &$new_object;
```

Dieser Code erzeugt zwei Referenzen auf dasselbe Objekt. Beachten Sie, dass `$another_object` als Referenz `$new_object` zugeordnet wird, nicht als Kopie hiervon. Wenn Sie versuchen die Referenzen zu kopieren, kopiert PHP nicht die Referenz, sondern erzeugt statt dessen eine Kopie der referenzierten Variablen. Nur wenn Sie jetzt Referenzen verwenden (um eine Referenz zu kopieren, müssen Sie wieder den Referenzoperator @ verwenden), können Sie sowohl `$new_object` als auch `$another_object` verwenden, um die Daten zu ändern, die sich innerhalb von `$my_object` befinden.

Für die unterschiedlichen Versionen geben wir einige Empfehlungen, auf die wir nachfolgend eingehen.

Klassen in PHP 3.0

Daten: Verwenden Sie Klassen nicht für komplexe Datenstrukturen, die echte Zeiger benötigen (wie etwa Bäume). Wenn es sich nicht vermeiden lässt, versuchen Sie die Verwendung von Klassen auf die Erfassung von Daten zu beschränken und die Datenverwaltung auszusparen.

Code: Verwenden Sie Klassen nur zur Strukturierung von APIs, die nicht durch die Kopiersyntax verletzt werden. Überlegen Sie, ob sich Ihr Projekt mit Prozeduren realisieren lässt. Wenn ja, erforschen Sie diese Möglichkeit. Prozeduren sind zuverlässig und haben sich bewährt, wohingegen Objekte noch einige Gefahren bergen.

Klassen in PHP 4.0

Verwenden Sie Klassen vorsichtig und stellen Sie sicher, dass Sie Kopien und Verweise von Objekten unterscheiden können. Achten Sie darauf, an welche Typen Sie Werte übergeben und wie Sie mit Ihnen umgehen. Sobald Sie nur ein einziges @ vergessen, erzeugt PHP eine Kopie Ihres Objekts, was wahrscheinlicher Weise die Konsistenz Ihrer Daten unterbrechen wird.

Die Klassen wurden zwar in PHP 4.0 verbessert, aber wir bleiben skeptisch. Wir haben Meinungen gehört, die in beide Richtungen gehen. Eine Gruppe besteht darauf, dass Objekte nichts taugen und nicht in einer Sprache wie PHP verwendet werden sollten, während eine andere Gruppe Objekte jedem prozeduralen Ansatz vorzieht, sogar in PHP 3.0. Für jene, die nur mit Prozeduren arbeiten, scheinen Objekte ein schlafendes Ungeheuer zu sein, das man besser nicht wecken sollte. Für die Fangemeinde der *objektorientierten Programmierung* sind die Anhänger des prozeduralen Ansatzes Idioten. Es ist fast wie ein Religionskrieg. Wann immer wir das Thema anschnitten, führte es zu endlosen und fruchtlosen Diskussionen.

Nach unserer Ansicht sind beide Extreme falsch. Es ist weder gut Funktionen komplett zu ignorieren noch sie ohne Betrachtung der Nachteile zu verwenden. Uns widerstrebt es aber auch, es als persönliche Vorliebe abzutun. Technologien sollten nie unter diesem Gesichtspunkt betrachtet werden. Unsere Empfehlung: Befreien Sie sich von jeglichen Vorurteilen, die Sie möglicherweise haben, insbesondere denjenigen von anderen, und entscheiden Sie dann objektiv, was für Ihr Projekt am besten geeignet ist.

2.4.2 Implementierung von Klassen

Ungeachtet der Pros und Contras sind Klassen ein wichtiges Element der Programmiersprache, und es scheint immer einen Bedarf zu geben, die Implementierung von Klassen in PHP zu erläutern.

Tatsächlich lassen sich Klassen in PHP recht einfach implementieren. Die meisten Schlüsselworte werden Sie bereits aus anderen Sprachen kennen:

```
class shopping_cart
{
    var $item_list;

    function pick($item, $quantity)
```

```
    {

        $this->item_list[$item] += $quantity;

    }

    function drop($item, $quantity)
    {

        if($this->item_list[$item] > $quantity)
            $this->item_list[$item] -= $quantity;
        else
            $this->item_list[$item] = 0;

    }

}
```

Dieser Code definiert die Klasse `shopping_cart` mit den Mitgliedern `$item_list`, `pick()` und `drop()`. Es gibt keine Möglichkeit öffentliche und private Mitglieder zu unterscheiden. In PHP wird standardmäßig alles auf öffentlich gesetzt – was bedeutet, dass Sie uneingeschränkt auf alle Elemente und Mitgliedsfunktionen einer Klasse von außen zugreifen können.

Dieses einfache Klassenbeispiel implementiert eine Art von Einkaufswagen (wie der Name sagt) mit einer Variablen, die den Inhalt des Wagens in einem assoziativen Array (`$item_list`) angibt, sowie zwei Funktionen (`pick()` und `drop()`), mit denen Einträge hinzugefügt und entfernt werden. Die Mitgliedsfunktionen sind wie reguläre Funktionen deklariert, mit der Ausnahme, dass sie in der Klassendefinition eingebettet sind. Die Klassenelemente (Variablen innerhalb der Klassen) werden mithilfe des Schlüsselworts »var« definiert.

> **Hinweis:** In PHP können Sie Klassendeklarationen und Implementierungen nicht trennen. Alle Funktionen müssen direkt in der Klassendeklaration implementiert werden.

2.4.3 Zugriff auf Objekte

Die Mitgliedsfunktionen können entweder über die »alte« Syntax `instance->member()` oder die »neue« Syntax `instance::member()` aufgerufen werden. Auf die gleiche Weise greifen Sie auf die Elemente entweder mit `instance->property` oder mit `instance::property` zu. Letztere Form eignet sich insbesondere dazu, den Konstruktor eines Elternknotens aufzurufen oder auf andere Mitglieder zuzugreifen, die sich nicht im aktuellen Objekt befinden (mehr zu diesem Thema im Abschnitt »Vererbung«):

```
class extended_cart extends shopping_cart
{

    function extended_cart()
    {

        shopping_cart::shopping_cart("Mousepad", 1);

    }

    function query($item)
    {

        return($this->item_list[$item]);

    }

}
```

Diese erweiterte Version des Objekts `extended_cart` besitzt einen Konstruktor, der den Konstruktor des Elternknotens aufruft, damit dieser den Rest den Objektbaums korrekt initialisiert. Würde der Konstruktor des Elternknotens nicht explizit von diesem Konstruktor aufgerufen, würde der Elternknoten nicht initialisiert werden (und damit auch nicht der Elternknoten dieses Knotens usw.).

Außerdem besitzt PHP einen Alias, der auf die aktuelle Instanz eines Objekts zeigt. Dieser Alias heißt `this` und gewährt Zugriff auf alle Mitglieder der aktuellen Instanz. Er wird für alle Selbstbezüge benötigt, da PHP innerhalb von Klassendefinitionen keinen lokalen Gültigkeitsbereich festlegt.

Hinweis: Da PHP keinen lokalen Gültigkeitsbereich innerhalb der Klassendefinitionen festlegt, stellen Sie sicher, dass Sie das Schlüsselwort `this` immer dann verwenden, wenn Sie innerhalb Ihres Objekts referenzieren. Wenn Sie `this` vergessen, referenziert PHP automatisch auf den globalen Gültigkeitsbereich, was recht fehlerträchtig ist!

2.4.4 Konstruktoren

Konstruktoren sind wie reguläre Funktionen definiert, mit der Ausnahme, dass ihre Namen mit den Klassennamen identisch sein müssen. Destruktoren kennt PHP nicht. Wie jede andere Funktion können auch Konstruktoren Argumente besitzen, selbst optionale Argumente sind zugelassen (weitere Informationen zu optionalen Parametern finden Sie weiter unten im Abschnitt über variable Argumentenlisten).

Hinweis: Seit PHP 4.0 sind für Kostruktoren nur skalare Werte als Parameter zugelassen (Zeichenketten, ganze Zahlen usw.), jedoch keine Arrays oder Objekte, wie es noch in PHP 3.0 der Fall war.

Um im vorangegangenen Beispiel einen Konstruktor hinzuzufügen, erweitern wir den Programmcode wie folgt:

```
class shopping_cart
{
    var $item_list;

    function shopping_cart($item = "T-Shirt", $quantity = 1)
    {

        $this->pick($item, $quantity);

    }

    function pick($item, $quantity)
    {

        $this->item_list[$item] += $quantity;

    }

    function drop($item, $quantity)
    {

        if($this->item_list[$item] > $quantity)
            $this->item_list[$item] -= $quantity;
        else
            $this->item_list[$item] = 0;

    }

}
```

Der Konstruktor, der in `shopping_cart::shopping_cart()` enthalten ist, hat zwei optionale Argumente. Wenn keine Argumente angegeben sind, wird der Einkaufswagen automatisch mit einem T-Shirt »vorgefüllt«. Ansonsten erhält er die gewünschten Einträge:

```
$default_cart = new shopping_cart;        // this cart will fill itself
                                             ➥ with one T-Shirt by default
$mug_cart = new shopping_cart("Mug", 2);  // this cart will contain two
                                             ➥ mugs
```

2.4.5 Vererbung

Um Funktionen zu Objekten hinzuzufügen sollte nicht der alte Code neu geschrieben, sondern existierende Strukturen stärker ausgenutzt werden. Neue Objekte können mithilfe des Schlüsselworts `extends` aus alten Objekten übernommen werden. Wie der Name sagt, wird hierdurch eine neue Klasse als Erweiterung einer bestehenden definiert:

```
class extended_cart extends shopping_cart
{

    function query($item)
    {

        return($this->item_list[$item]);

    }

}
```

Dieser erweiterte Wagen `extended_cart` enthält nun alle Elemente und Mitgliedsfunktionen von `shopping_cart` sowie zusätzlich eine weitere Funktion `query()`, mit welcher wir die Menge eines bestimmten Eintrags im Wagen überprüfen können.

Hinweis: Die Klasse `extended_cart` besitzt keinen eigenen Konstruktor. Wenn die Klasse eines Tochterknotens keinen Konstruktor hat, ruft PHP (seit der Version 4.0) automatisch den Konstruktor des Elternknotens auf. Standardmäßig wird PHP jedoch nie den Konstruktor eines Elternknotens aufrufen. Wenn Sie also das übergeordnete Objekt einrichten müssen, stellen Sie sicher, dass Sie seinen Konstruktor manuell aufrufen.

2.4.6 Spezielle Funktionen für objektorientierte Programmierung

PHP hat ein paar schöne Funktionen, welche den Umgang mit Objekten erleichtern. Die folgende Tabelle beschreibt diese Funktionen.

Funktion	Beschreibung
`string get_class (object object)`	Gibt den Namen der angegebenen Objektinstanz als Zeichenkette zurück.
`string get_parent_class (object object)`	Gibt den Namen der übergeordneten Klasse der angegebenen Objektinstanz als String zurück.

Funktion	Beschreibung
bool method_exists (object object, string method)	Prüft, ob die in method genannte Funktion tatsächlich ein Mitglied von object ist.
bool class_exists (string classname)	Prüft, ob classname eine tatsächlich definierte Klasse ist.
bool is_subclass_of (object object, string classname)	Prüft, ob object eine Unterklasse von classname ist.

Hinweis: Diese Funktionen sind nicht in PHP 3.0 verfügbar.

Quellcode für den »Einkaufswagen«

Dieser Abschnitt erläutert anhand des »Einkaufswagens« die vollständige Implementierung eines objektorientierten Programms (siehe Listing 2.1). Das Beispiel ist zwar sehr einfach, aber trotzdem nützlich.

```
class shopping_cart
{

    var $item_list;

    function shopping_cart($item = "T-Shirt", $quantity = 1)
    {

        $this->pick($item, $quantity);

    }

    function pick($item, $quantity)
    {

        $this->item_list[$item] += $quantity;

    }

    function drop($item, $quantity)
    {

        if($this->item_list[$item] > $quantity)
            $this->item_list[$item] -= $quantity;
        else
            $this->item_list[$item] = 0;

    }
```

```
}

class extended_cart extends shopping_cart
{

    function query($item)
    {

        return($this->item_list[$item]);

    }

    function get_contents()
    {

        return($this->item_list);

    }

}

// you can instantiate shopping_cart the regular way
$cart = new shopping_cart;

// you can make use of the variable arguments of the constructor
$cart = new shopping_cart("Cap", 2);

// you can also use extended_cart, which in turn calls the
// constructor of shopping_cart implicitly
$cart = new extended_cart;

// or you can use the inherited features of the constructor
$cart = new extended_cart("Cap", 2);

// of course, you can also use the inherited functions
$cart->pick("Mug", 1);

// ...or use any functions in the object itself
while(list($item, $quantity) = each($cart->get_contents()))
    print("We have $quantity of $item");
```

Listing 2.1: Einkaufswagen-Quellcode

2.5 Verknüpfte Listen

Verknüpfte Listen, eine Sonderform der Baumstrukturen, gehören zu den typischen Datenstrukturen für die Organisation von dynamischen Datensätzen. Wir gehen davon aus, dass Sie bereits ein Grundverständnis von der Struktur, dem Konzept und der Verwendung verknüpfter Listen haben. Daher werden wir hier nicht detailliert auf ihre Implementierung eingehen, sondern konzentrieren uns auf die wichtigsten zu beachtenden Aspekte.

Wie bereits in den vorangegangenen Abschnitten beschrieben, erzeugt PHP 3.0. von den Objektinstanzen Kopien, anstatt diese mithilfe eines Zeigers zu referenzieren. Damit ist nur eine sehr begrenzte, WORM-ähnliche Verwendung der verknüpften Listen möglich: einmal schreiben, mehrfach lesen. Verknüpfte Listen können erzeugt, aber nicht verändert werden. Wenn Sie versuchen, ein Element in der Liste zu ändern, verlieren Sie die Referenz auf alle folgenden Elemente in der Liste. Aus demselben Grund lässt sich auch die Reihenfolge der Elemente nicht verändern.

Doppelt verknüpfte Listen lassen sich in PHP 3.0 ebenfalls nicht realisieren (zumindest haben wir es nicht geschafft, wobei wir einige Stunden mit Fehlersuche und Behebung zubrachten, bevor wir aufgegeben haben). Da jeder Knoten eine neue Kopie des Listenendes benötigte, mit dem es verknüpft ist, müssten Sie eine Vielzahl redundanter Listen mit demselben Inhalt erstellen, nur um die Funktion »ein Element zurückgehen« zu ermöglichen.

PHP 4.0, das echte Referenzen unterstützt, hat diese Einschränkungen nicht. Listen können beliebig erzeugt und neu zusammengestellt werden, selbst doppelt verknüpfte Listen. Beachten Sie jedoch, dass es nicht ganz einfach ist, zwischen Referenzen und den tatsächlichen Kopien von Listenelementen zu unterscheiden.

Ein Motto von Programmierern konventioneller Programmiersprachen lautet: Hüte dich vor freien Zeigern. Wir möchten dieses für PHP modifizieren: Hüte dich vor redundanten Kopien.

Wenn Sie mit Listen arbeiten, erstellen Sie eine absturzsichere Bibliothek, die auf möglichst allgemeine Weise Ihre Daten nach Ihren Bedürfnissen verwaltet. Testen Sie die Bibliothek ausführlich und stellen Sie sicher, dass sie korrekt arbeitet. Das erspart Ihnen, fehlerhaften Programmcode zu suchen, der auf Ihre Listen auf falsche Weise zugreift und diese möglicherweise zerstört.

2.5.1 Verknüpfte Listen und Baumstrukturen – ein Arbeitsansatz

Wie bereits erwähnt, empfiehlt es sich, für Ihre Bedürfnisse eine absturzsichere Bibliothek zu erstellen – eine, die sich einfach erweitern lässt und alle für die Aufgabe erforderlichen Informationen bereitstellt. Im Folgenden

möchten wir ein Beispiel aus der Praxis vorstellen: eine Bibliothek, die wir zur Verwaltung von Baumstrukturen entwickelt haben, und die auch mit PHP 3.0 arbeitet. Den vollständigen Quellcode finden Sie auf der CD-ROM.

Die Bibliothek kann doppelt verknüpfte Baumstrukturen mit zwei Tochterknoten pro Blattknoten handhaben, wobei jeder Knoten einen Inhaltsbehälter für gemischte Variablen hat. Jede Aktion, die sich mit der Baumstruktur durchführen lässt, wurde in die Anwendungsprogrammierschnittstelle eingebaut. Sie erhalten zwei separate Einheiten: die Baumstruktur und den auf sie zugreifenden Programmcode.

Dies ist genau der Grund, warum diese Baumstruktur mit PHP 3.0 funktioniert (auch wenn dies scheinbar in Widerspruch zu dem zuvor Gesagten steht). Die Baumstruktur basiert nicht auf Zeigern, sondern auf Arrays. Da PHP dynamische Arrays unterstützt, ist es mit einem geringen Aufwand möglich, diese Arrays zu Erstellung einer dynamischen Baumstruktur zu verwenden.

Die Idee ist nicht neu. Sie existiert bereits seit einigen Jahren und ist nicht schwer zu verstehen. Anstatt Zeiger zu verwenden, welche auf die Speicheradresse eines anderen Knotens verweisen, mit welchem ein Knoten verknüpft ist, sind im Array alle Knoten mit den Indizes der Knoten versehen, mit denen sie verknüpft sind. Dies hat nicht nur den Vorteil, dass PHP bei Verwendung von ungültigen Indizes eine Warnung ausgibt, Sie können auch den gesamten Baum kopieren, indem Sie die Variable, die das Array eines Baumes angibt, einer anderen Variablen zuweisen. Darüber hinaus können Sie die gesamte Baumstruktur, so wie sie ist, in eine Datei kopieren und diese an eine beliebige andere Stelle verschieben.

Eine weitere, eher theoretische Erläuterung, ist folgende: Stellen Sie sich den für Ihr Programm verfügbaren Speicher als ein großes Array vor. Die Elemente wären im Falle von physischem RAM wahrscheinlich einige Byte groß, aber die Größe der einzelnen Elemente ist nicht wirklich von Bedeutung. Wichtiger ist, dass der Zeiger einfach eine Nummer ist, die eines der Elemente indiziert und damit den Beginn jeder Struktur markiert, die Sie in Ihr RAM einfügen. Wenn Sie nun das gesamte Konstrukt in eine Sprache integrieren (ein »echtes« Array), haben Sie dieselbe Situation auf einer höheren Ebene. Das PHP-Array enthält nur Ihr »RAM«, und jedes Array-Element stellt einen der drei Baumknoten dar. Die Zeiger werden in diesem Array zu Indizes, und die Referenzierung erfolgt einfach durch Auslesen des korrekten Elements aus dem Array.

Mithilfe von Arrays können Sie viele »RAMs« erzeugen, ihre Größe erhöhen oder verringern, über sie als Ganzes oder nur ein einzelnes Element verfügen; alles in allem eine sehr komfortable Methode zur Speicherverwaltung. Wenn Sie dies alles in einer soliden Bibliothek integriert haben, besitzen Sie ein schönes Werkzeug.

Abbildung 2.9 zeigt, wie die Baumbibliothek die Baumknoten in einem Array intern verwaltet.

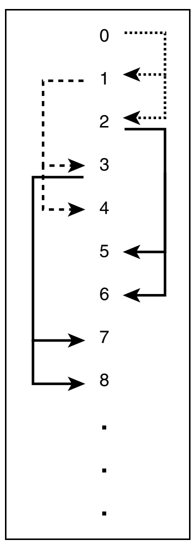

Abbildung 2.9: Array, das eine Baumstruktur enthält

Die Bibliothek besteht aus den folgenden Funktionen:

Funktion	Beschreibung
`array tree_create()`	Erzeugt einen neuen Baum.
`int tree_allocate_node (array tree)`	Ordnet in der Baumstruktur einen neuen Knoten zu.
`int tree_free_node (array tree, int handle)`	Gibt einen Knoten in der Baumstruktur frei.
`int tree_link_left (array tree, int link_to, int child)`	Verknüpft einen Knoten als linken Tochterknoten mit einem anderen Knoten.
`int tree_link_right (array tree, int link_to, int child)`	Verknüpft einen Knoten als rechten Tochterknoten mit einem anderen Knoten.
`int tree_get_parent (array tree, int handle)`	Gibt den Elternknoten des angegebenen Knoten zurück.
`int tree_get_left (array tree, int handle)`	Gibt den linken Tochterknoten des angegebenen Knotens zurück.
`int tree_get_right (array tree, int handle)`	Gibt den rechten Tochterknoten des angegebenen Knotens zurück.

Funktion	Beschreibung
`int tree_assign_node_contents (array tree, int handle, mixed contents)`	Weist dem Inhaltsbehälter eines Knotens Daten zu.
`mixed tree_retrieve_node_contents (array tree, int handle)`	Liest Daten aus dem Inhaltsbehälter eines Knotens aus.

Wenn Sie möchten, können Sie hier weitere Funktionen hinzufügen. Die Bibliothek hat beispielsweise noch keine Funktionen um Bäume zu vermischen, tote Knoten zu entdecken (Knoten, die zwar im Array vorhanden, aber vom Hauptbaum abgeschnitten sind und auf die somit nicht mehr zugegriffen werden kann) usw. Dieser Code eignet sich hervorragend zum Experimentieren und Lernen.

Die Bibliothek ist recht einfach. `tree_create()` erzeugt ein neues Array für die Baumstruktur und initialisiert das erste Element als Stammelement. Alle Referenzen auf andere Knoten sind ganzzahlige Indizes innerhalb des Arrays (siehe `$idx_up`, `$idx_left` und `$idx_right` in der Quelle). -1 markiert eine gerade verwendete Referenz. Wenn ein Knoten beispielsweise keinen linken Tochterknoten hat, enthielte `$idx_left` die Angabe -1. Um zu markieren, ob ein Element selbst in Gebrauch ist oder nicht (d.h. ob ihm Daten zugewiesen sind), wird ein anderer Flag definiert: `$free`. Diese Variable hat entweder den Wert `1` (in Gebrauch) oder `0` (nicht in Gebrauch).

`tree_create()` erzeugt einen Dummyknoten und markiert ihn als frei, d.h. alle Referenzen werden nicht verwendet, und weist ihn dem Slot 0 in dem Baum-Array zu. Anschließend wird dieses Array an den Anrufer zurückgeschickt.

Hinweis: Der Anrufer muss nichts über dieses Array wissen, nicht einmal dass es ein Array ist. Das Programm sollte es einfach für etwas als Baumstruktur »zu bearbeitendes« halten. Da PHP keine expliziten Typen hat und verlangt, funktioniert dies sehr gut.

`tree_allocate_node()` sucht innerhalb des Baum-Arrays nach einem freien Knoten, indem es für jeden bestehenden Knoten den Flag `$free` überprüft. Wenn keiner der Knoten als frei markiert ist, weist es einfach einen neuen zu und fügt es dem Baum-Array hinzu. Hier macht sich die dynamische Struktur von PHP bemerkbar: Wenn wir Arrays fester Größe nutzen müssten, gingen uns früher oder später die Knoten aus. Der gefundene Knoten wird dann als »in Gebrauch« markiert und als »zu bearbeiten« an den Anrufer zurückgeschickt.

`tree_free_node()` macht genau das Gegenteil. Es markiert den angegebenen Knoten als »nicht in Gebrauch«, indem es den Flag `$free` löscht. Dies wirft drei Probleme auf: Zunächst sind freie Knoten nicht wirklich frei, sie sind nur als frei markiert. Angenommen, Sie wollten eine komplexe Baumstruktur mit vielen Knoten erstellen und anschließend ein Optimierungsprogramm darüber laufen lassen, das in der Regel die Anzahl der Knoten um die Hälfte reduziert und viele Geisterknoten im Array zurücklässt. Wenn wir davon ausgehen, dass Sie ursprünglich 1.000 Knoten zugeordnet haben und während der Optimierung 500 von ihnen freisetzen, erhalten Sie ein Array, das immer noch 1.000 Knoten enthält, wobei hierbei nur 500 von ihnen als »in Gebrauch« markiert sind. Dies ist eine ziemliche Verschwendung von Speicherplatz. Eine automatische Speicherplatzbereinigung wäre hier sicherlich eine gute Idee.

Speicherbereinigung wirft jedoch ein zweites Problem auf, nämlich Zombieknoten. Zombieknoten sind Knoten, die als »in Gebrauch« markiert sind, aber keine Verknüpfung mehr zum Baum haben und daher nicht mehr referenziert werden können (siehe Abbildung 2.10).

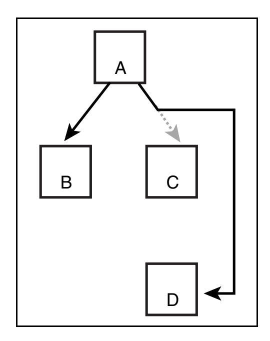

Abbildung 2.10: Zombieknoten in einer Baumstruktur

Da die Bibliothek alle Informationen über die Knoten in der Baumstruktur und ihre interne Verknüpfung enthält, lassen sich Zombieknoten sehr einfach ausmachen. Dies fehlt jedoch leider noch im Programmcode.

Das dritte Problem ähnelt dem der Zombieknoten: unterbrochene Verknüpfungen. Unterbrochene Verknüpfungen sind Verknüpfungen, die von einem Knoten ausgehen und auf einen nicht verwendeten bzw. nicht existenten Knoten zeigen (siehe Abbildung 2.11). Die Verknüpfungen werden »unterbrochen«, sobald Sie einen Knoten von dem Baum abtrennen, ihn als »frei« markieren bevor Sie alle anderen Knoten geändert haben, die auf diesen Knoten referenzieren.

Auch dies lässt sich durch eine strenge Überprüfung der Bibliotheksfunktionen und die Speicherbereinigung beseitigen.

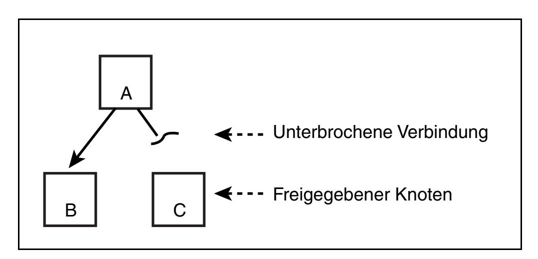

Abbildung 2.11: Unterbrochene Verknüpfungen in einer Baumstruktur

`tree_link_left()` und `tree_link_right()` verknüpfen einen linken bzw. rechten Tochterknoten, indem sie den zugehörigen Handle den Eigenschaften `$idx_left` und `$idx_right` in der Knotenstruktur zuordnen. Sie finden die Gegenparts in `tree_get_left()` und `tree_get_right()`. Diese lesen den Handle der linken bzw. rechten Verknüpfung des Knotens aus. Zusätzlich ermittelt `tree_get_parent()` den Elternknoten eines Unterknotens.

Um den Inhalt in der Baumstruktur zu speichern und Daten aus ihr auszulesen, können Sie `tree_assign_node_contents()` und `tree_retrieve_node_contents()` verwenden. Auch hier ist der dynamische Charakter von PHP hilfreich, da wir unsere Knoten nicht auf bestimmte Datentypen festlegen müssen. Es ist inzwischen in C++ allgemein üblich, Instanzen mithilfe von Klassenvorlagen zu erzeugen, die z.B. Baumklassen nur für ganze Zahlen generieren. Sie könnten zwar unzählige Baumstrukturen für eine beliebige Anzahl von Datentypen entwerfen, doch dynamisch eingegebene Inhalte können mit dieser Methode nur schwer gespeichert werden. PHP akzeptiert gemischte Typen problemlos, wodurch Sie alle Knotendatentypen jederzeit ändern können.

Übung: Fügen Sie eine zuverlässige Speicherbereinigung in Ihre Baumbibliothek ein.

Hinweis: Sie können in der Knotenstruktur einen neuen Flag für die Referenzzählung einführen. Dies erleichtert und beschleunigt die Speicherbereinigung.

Unterschätzen Sie die Übungen nicht. Zu wissen, wie etwas funktioniert, ist etwas ganz Anderes, als in der Lage zu sein, etwas zu tun. Wir empfehlen Ihnen wärmstens, zumindest zu versuchen, die Bibliothek zu verbessern. Es ist absolut keine Zeitverschwendung, selbst wenn Sie keinen Erfolg haben. Um es mit den Worten Marie Freifrau von Ebner-Eschenbach zu sagen: »Für das Können gibt es nur einen Beweis: das Tun.«

Listing 2.2 zeigt die vollständige Implementierung der Baumbibliothek:

```
//
// This structure keeps a tree node
//
class tree_node
{
    // array indices linking to neighboring nodes
    var $idx_up;
    var $idx_left;
    var $idx_right;

    var $free;

    // contents of this node, this is a mixed variable
    var $contents;
}

function tree_create()
{

    // create a new, empty array
    $return_array = array();

    // allocate the root node
    $root_node = new tree_node;

    // all other linking indices are invalid
    $root_node->idx_up = -1;
    $root_node->idx_left = -1;
    $root_node->idx_right = -1;

    // this node is unused
    $root_node->free = 1;

    // create dummy contents
    $root_node->contents = "";

    // assign root element to array
    $return_array[0] = $root_node;

    // return it back to caller
    return($return_array);

}

function tree_allocate_node(&$tree_array)
```

```
{
    // find a free node
    for($i = 0; $i < count($tree_array); $i++)
    {
        // retrieve node from array
        $node = $tree_array[$i];

        // is it in use?
        if($node->free)
        {
            // no, it is not in use, allocate it
            $node->free = 0;

            // assign node back to array to update the tree
            $tree_array[$i] = $node;

            // now return this node's index as handle
            return($i);
        }
    }

    // we haven't found a free node, so allocate a new one
    $node = new tree_node;

    // invalidate all indices
    $node->idx_up = -1;
    $node->idx_left = -1;
    $node->idx_right = -1;

    // this node is NOT free
    $node->free = 0;

    // assign dummy contents
    $node->contents = "";

    // now add this node to the tree array
    $tree_array[] = $node;

    // return new index as handle
    return(count($tree_array) - 1);

}

function tree_free_node(&$tree_array, $handle)
```

```
{

    // retrieve node from tree
    $node = $tree_array[$handle];

    // check if it is really allocated
    if($node->free)
        // this node is free, return an error code
        // note that this only serves diagnostic
        // purposes since it wouldn't hurt the tree
        // if we'd just mark it as free
        return(-1);

    $node->free = 1;

    // assign node back to tree
    $tree_array[$handle] = $node;

    return(1);

}

function tree_link_left(&$tree_array, $link_to, $child)
{

    // retrieve nodes
    $link_node = $tree_array[$link_to];
    $child_node = $tree_array[$child];

    // check if nodes are allocated
    if($link_node->free || $child_node->free)
        // return error, we do not allow linkage
        // of free nodes
        return(-1);

    // link nodes together
    $link_node->idx_left = $child;
    $child_node->idx_up = $link_to;

    // write nodes back into the array
    $tree_array[$link_to] = $link_node;
    $tree_array[$child] = $child_node;

    // return success
    return(1);
```

```
}

function tree_link_right(&$tree_array, $link_to, $child)
{

    // retrieve nodes
    $link_node = $tree_array[$link_to];
    $child_node = $tree_array[$child];

    // check if nodes are allocated
    if($link_node->free || $child_node->free)
        // return error, we do not allow linkage
        // of free nodes
        return(-1);

    // link nodes together
    $link_node->idx_right = $child;
    $child_node->idx_up = $link_to;

    // write nodes back into the array
    $tree_array[$link_to] = $link_node;
    $tree_array[$child] = $child_node;

    // return success
    return(1);

}

function tree_get_parent(&$tree_array, $handle)
{

    // retrieve node from array
    $node = $tree_array[$handle];

    // check if node is actually allocated
    if($node->free)
        // node is not allocated, return error
        return(-1);

    // node is allocated, return its parent
    return($node->up);

}

function tree_get_left(&$tree_array, $handle)
{
```

```
    // retrieve node from array
    $node = $tree_array[$handle];

    // check if node is actually allocated
    if($node->free)
        // node is not allocated, return error
        return(-1);

    // node is allocated, return its left child
    return($node->left);

}

function tree_get_right(&$tree_array, $handle)
{

    // retrieve node from array
    $node = $tree_array[$handle];

    // check if node is actually allocated
    if($node->free)
        // node is not allocated, return error
        return(-1);

    // node is allocated, return its left child
    return($node->right);

}

function tree_assign_node_contents(&$tree_array, $handle, $contents)
{

    // retrieve node from array
    $node = $tree_array[$handle];

    // check if node is actually allocated
    if($node->free)
        // node is not allocated, return error
        return(-1);

    // assign contents to node
    $node->contents = $contents;

    // assign node back into array
    $tree_array[$handle] = $node;
```

```
    // return success
    return(1);

}

function tree_retrieve_node_contents(&$tree_array, $handle)
{

    // retrieve node from array
    $node = $tree_array[$handle];

    // check if node is actually allocated
    if($node->free)
        // node is not allocated, return error
        return(-1);

    // return contents of this node
    return($node->contents);

}
```

Listing 2.2: Implementierung der Baumbibliothek

2.6 Assoziative Arrays

Eine weitere grundlegende Struktur der Programmiersprachen stellen Arrays dar. Sie bieten die Möglichkeit, einen festen Satz (oder eine Sammlung) von Daten desselben Datentyps zu speichern, wodurch die Elemente in ihrem Datensatz über einen eindeutigen Schlüssel indexierbar werden.

In typischen »konventionellen« Programmiersprachen werden Arrays auf folgende Weise verwaltet:

```
int my_integer_array[256];          // allocate 256 integers in this array
```

Dieser Ausschnitt eines C-Programmcodes deklariert ein Arrray namens `my_integer_array`, das 256 ganze Zahlen enthält. Sie können jede dieser ganzen Zahlen adressieren, indem Sie das Array mit einer Ordnungszahl indexieren, wobei die Zahl in diesem Fall zwischen 0 und 255 liegen muss. (C beginnt die Zählung bei 0; die Zahl in der Array-Definition gibt die Anzahl der ganzen Zahlen an, die verfügbar sein sollen.) Die Indexierung erfolgt folgendermaßen:

```
int my_integer = my_integer_array[4];
```

Hierdurch wird das fünfte Element (denken Sie daran, C beginnt die Zählung bei 0) aus dem Array ausgelesen und in `my_integer` gespeichert.

Aufgrund des Wesens kompilierter Sprachen waren Sie stets an die vorausgegangene Definition Ihrer Variablen gebunden. Wenn Sie im obigen Array plötzlich mehr als 256 ganze Zahlen benötigen, hätten Sie leider Pech. Natürlich hätten Sie diese Variable als Zeiger auf ein ganzzahliges Array definieren und diesem 257 Elemente zuweisen können. Aber was wäre, wenn Sie plötzlich ein weiteres Element benötigten? Sie müssten einen neuen Speicherbereich zuweisen, den alten Inhalt des Arrays kopieren und den alten, nunmehr nicht mehr verwendeten Speicherbereich freigeben.

PHP verfolgt einen anderen Ansatz. Da PHP keine typischen Variablendeklarationen kennt (nur Typdefinitionen), werden neue Variablen quasi im Vorübergehen zugewiesen. Sobald Sie eine neue Variable generieren, indem Sie ihren Namen in den Namensraum einfügen, schaffen Sie einfach Speicherplatz, der an diesen Namen gebunden ist, nichts weiter. Die Art der Daten, die in diesem Speicherbereich abgelegt werden, ist nicht auf einen bestimmten Variablentyp beschränkt. Die Datentypen können jederzeit neu interpretiert werden, ihre Größe kann verändert werden, sie können neu zugewiesen werden und vieles mehr.

Betrachten wir uns folgendes Beispiel:

```
$my_var = 1;
$my_var = "Used to be an integer";
$my_var = array("Oh well, I like arrays better");
```

Die erste Zeile erzeugt eine neue Variable `$my_var`. PHP wird feststellen, dass ihr eine ganze Zahl zugeordnet wird und setzt daher den ursprünglichen Typ von `$my_var` auf ganze Zahl. Die zweite Zeile überschreibt den Inhalt von `$my_var` mit einer Zeichenkette. Bei einer konventionellen Programmiersprache würde dieses bei der Kompilierung zu einem Fehler führen oder zumindest zu einem Programmabbruch während der Laufzeit. PHP ändert den Typ von `$my_var` jedoch dynamisch auf »string« und ordnet die Variable neu zu, so dass genügend Speicherplatz für die Zeichenkette zur Verfügung steht. Die Zeile ändert dann den Typ von `$my_var` erneut, indem es ein Array davon erzeugt. PHP meistert alle Fälle transparent und problemlos. (Wir wissen, dass es da draußen noch weitere Sprachen ohne strikte Variablentypen gibt, aber wir möchten diese hier nicht als konventionelle Sprachen klassifizieren.)

Hinweis: PHP 3.0 hat keine vernünftige Speicherbereinigung. Bei der Neuzuweisung einer Variablen wird der bereits zugewiesene Speicherplatz nicht immer wieder verwendet. Bei langfristigen Skripten (oder Skripten, die eine umfangreiche Verarbeitung haben) könnte dies zu größeren Bereichen »toten Speichers« führen. Bei der Verwendung von speicherintensiven Skripten, die für einen langen Zeitraum genutzt werden, sollten Sie die Speicherbelegung in einer Testumgebung überwachen, bevor Sie die entsprechenden Skripte in einer produktiven Umgebung implementieren, um sicherzugehen, dass Ihr Server nicht überlastet wird. PHP 4.0 ist nicht so anfällig für dieses Problem.

Da PHP keine formalen Variablendeklarationen benötigt, ist die Verwendung der Variablen vollständig dynamisch. Ein Sonderfall der dynamischen Variablenverarbeitung in PHP sind die Arrays. Wahrscheinlich kennen Sie den herkömmlichen Array-Typ, das indexierte Array. Indexierte Arrays sind Arrays, die durch Ordnungszahlen indiziert sind. Diese Ordnungszahlen gehen in der Regel von 0 bis n, wobei n der höchstmögliche Index ist. Sprachen wie Pascal lassen eine Indexierung mit verschiedenen Bereichen, wie etwa von 3 bis 18 zu, wobei diese Bereiche später zur Laufzeit auf 0-basierte Indizes zurückgewandelt wird. Das Hauptmerkmal dieser indexierten Arrays ist, dass sie andere Indizes von einem bestimmtem Basisindex berechnen können. Nehmen wir beispielsweise an, Sie möchten bei Index 2 beginnend drei aufeinanderfolgende Array-Elemente auslesen:

```
$base_index = 2;

for($i = $base_index; $i < $base_index + 3; $i++)
    print("Element $i is $my_array[$i]<br>");
```

Bei jeder Iteration der `for()`-Anweisung berechnet dieser kleine Programmteil den nächsten Index in dem Array durch Heraufsetzen von `$i`.

Assoziative Arrays haben diese Funktion nicht. Die Besonderheit bei assoziativen Arrays besteht darin, dass Sie mit Nicht-Ordnungszahlen wie etwa Zeichenketten indiziert werden können. Jeder Zeichenkette, die als Index verwendet wird, ist ein Wert zugewiesen, was auch den Namen »assoziatives Array« erklärt. Wie Sie sich vielleicht denken können, lässt sich auch durch Zuteilen eines Basisindexes der nächste gültige Index in dem Array nicht erraten. Sie müssen die Array-Schlüssel kennen, um die zugehörigen Werte auszulesen.

Daneben können die bereits zuvor besprochenen Funktionen `list()` und `each()` verwendet werden, um assoziative Arrays zu verarbeiten.

Indizierte Arrays sind in PHP nur eine Sonderform der assoziativen Arrays. Wenn Sie `unset()` auf eines der Elemente in einem indexierten Array anwenden, bleiben alle anderen Elemente (und ihre Reihenfolge) unberührt, jedoch entsteht ein unzusammenhängendes Array. Einzelheiten hierzu finden Sie in den bereits besprochenen Abschnitten über `list()` und `each()`.

2.6.1 Mehrdimensionale Arrays

Wie der Name vermuten lässt, besitzen mehrdimensionale Arrays mehr als eine Dimension. Eindimensionale Arrays haben die folgende Form, die für Arrays wohl am bekanntesten ist:

```
$my_array[0] = 1;
$my_array[1] = 777;
$my_array[2] = 45;
```

Um diesem Array-Typ zu indizieren, brauchen Sie nur einen Index, der die Anzahl der möglichen Werte auf den Bereich dieses Index begrenzt. Häufig ist es jedoch sinnvoll, mehrdimensionale Arrays zu erzeugen, wenn Sie komplexe Datensätze verwalten. Typische Beispiele hierfür sind Bitmaps und Bildschirmpuffer. Wenn Sie auf Ihren Bildschirm blicken, sehen Sie (zumindest heutzutage) eine zweidimensionale Projektion Ihres Desktops. Die Fenster, Bitmaps, Befehlszeilen, Cursor, Zeiger, alles ist zweidimensional. Um diese Daten auf bequeme Weise darzustellen, könnten Sie natürlich alles in Arrays mit einer einzigen Dimension zusammenfassen. Es ist jedoch passender, Arrays zu verwenden, welche dieselbe Anzahl von Dimensionen haben wie die Eingabedaten. Um beispielsweise eine Bitmap (eine Reihe von Pixel) für einen Mauszeiger zu speichern, könnten Sie in Ihrem Array einen weiteren Index hinzufügen:

```
// clear mouse bitmap
for($x = 0; $x < MOUSE_X_SIZE; $x++)
    for($y = 0; $y < MOUSE_Y_SIZE; $y++)
        $mouse_bitmap[$x][$y] = 0;
```

Hiermit wird eine Maus-Bitmap gelöscht und alle Elemente auf 0 gesetzt. Dabei werden zwei Schleifen verwendet, eine für jede Dimension. Abbildung 2.12 zeigt eine grafische Darstellung von zweidimensionalen Daten mit Daten aus einem Koordinatensystem. Intern, d.h. im Speicher, werden die Datenelemente natürlich in Serie gespeichert (der RAM hat nur eine Dimension für die Indizierung). Eine passende Analogie für die Visualisierung bietet das Koordinatensystem.

In Arrays gibt es keine Begrenzung für die maximale Anzahl der Dimensionen (offen gesagt haben wir es noch nicht ausprobiert, aber es wird kaum eine Anwendung für Arrays mit 16 Dimensionen oder mehr geben). Dimensionen

können auch in verschiedenen Typen auftreten (die erste Dimension assoziativ, die zweite Dimensionen mit ganzzahligen Indizes, die dritte wieder assoziativ usw.). Sie eignen sich damit auch hervorragend für die Darstellung statistischer Daten.

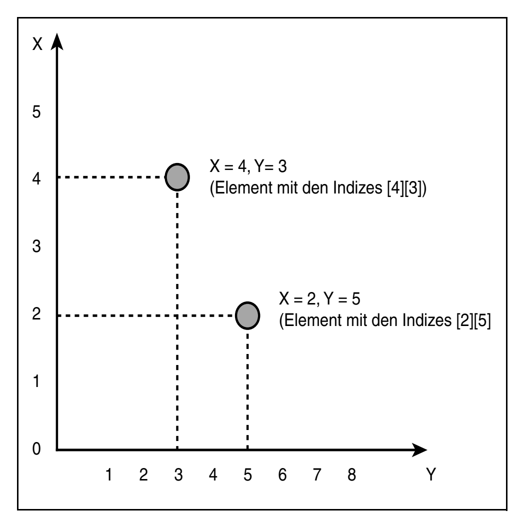

Abbildung 2.12: Eine zweidimensionale Array-Struktur

2.6.2 Variable Argumente

Bei der Verwendung von Funktionen ist es häufig notwendig, mehr als einen Wert zurückzugeben oder die gegebenen Parameter zu ändern. `fsockopen()` gibt beispielsweise als Rückgabewert den Sockel-Bezeichner zurück, denkbar wäre jedoch auch die Ausgabe eines Fehlercodes mit beschreibendem Text, z.B. bei potentiellen Fehlern:

```
// try to open a socket for HTTP with a 30 second timeout
$socket_handle = fsockopen("www.myhost.com", 80, $error_nr, $error_txt,
30);
if(!$socket_handle)
{
    print("Couldn't connect to HTTP host.<br>");
    print("Error code: $error_nr, Reason: $error_txt<br>");
}
```

Wenn eine Verbindung zum gewünschten Host nicht errichtet werden konnte, gibt dieser Code einen Fehlercode und den Fehlergrund aus. Die Variablen wurden ursprünglich als Parameter an `fsockopen()` übergeben. Da diese Parameter in der `fsockopen()`-Deklaration jedoch als »durch Referenz weitergegeben« deklariert sind, kann `fsockopen()` sie modifizieren und die Änderungen anschließend global zur Verfügung stellen.

In der Regel greifen Funktionen auf ihre Parameter nicht durch Referenz zu. Wenn sie ihre Werte während der Programmausführung ändern, arbeiten sie mit einer lokalen Kopie des ursprünglichen Wertes.

```
function calculate($a, $b, $c)
{

    $a = $b + $c;

}

$i = 1;
$j = 2;
$k = 3;

print("I $i, J $j, K $k<br>");
calculate($i, $j, $k);
print("I $i, J $j, K $k<br>");
```

Beide Druckanweisungen geben denselben Inhalt aus. Während `calculate()` bei der Programmausführung $a modifiziert, ändert sich der Inhalt von $i nicht, obwohl er als Argument für den Parameter $a übergeben wurde. Dies liegt daran, dass `calculate()` nicht mit der Variablen selbst, sondern mit einer Kopie der ursprünglichen Variablen arbeitet. Sobald die Funktion beendet ist, wird die Kopie der Variablen verworfen, mit welcher die Anweisung gearbeitet hat, und der Inhalt geht verloren.

Wie im Falle von `fsockopen()` kann es manchmal wünschenswert sein, die an einem Parameter vorgenommene Änderung zu speichern und sie global zur Verfügung zu stellen. Damit dies möglich wird, muss die Variable jedoch nicht als Kopie sondern als Referenz übergeben werden. Damit erhält die Funktion einen Zeiger auf den Speicherblock, in dem die ursprüngliche Variable abgelegt ist. Mithilfe dieses Zeigers kann die Funktion auf die globale Instanz der Variablen zugreifen und diese direkt ändern:

```
function calculate(&$a, $b, $c)
{

    $a = $b + $c;

}

$i = 1;
$j = 2;
$k = 3;

print("I $i, J $j, K $k<br>");
calculate($i, $j, $k);
print("I $i, J $j, K $k<br>");
```

Wie Sie hier sehen, ist hier nur ein Zeichen anders: das Ampersand (&), das im vorherigen Programmcode fehlte. Dieses, einem Funktionsparameter vorangestellte Zeichen gibt an, dass er per Referenz übergeben wurde.

Hinweis: Um ein Ampersand (&) einzufügen, müssen Sie nicht die Zeile ändern, in welcher die Funktion aufgerufen wird. PHP wandelt Ihre Parameter automatisch in Referenzen um, wenn es eine Funktion findet, die Parameter auf diese Weise übergeben möchte.

Bei der Änderung von $a durch `calculate()` wird die lokale Kopie von $i, demnach nicht verändert, statt dessen wird durch den Zugriff auf den globale Speicher von $i die Änderung direkt durchgeführt. Viel wichtiger ist aber, dass die Änderungen, die an $i vorgenommen wurden, bei Beendigung der Funktion nicht verloren gehen, was beim Arbeiten mit einem lokalen Speicher nicht funktioniert.

Die Parameterübergabe durch Referenz ist häufig eine nützliche Methode, wenn Sie mit einer Funktion mehr als einen Wert ausgeben möchten oder Ihre Variablen durch mehrfachen Aufruf einer Funktion auf einmal ändern wollen, um eine komplexe Berechnung in einem Durchgang durchzuführen. Sie sollten es jedoch vermeiden, Strukturen aufzusplitten und in Parameterlisten zu zwängen, wenn Sie die Daten genauso gut direkt in einer Struktur zurückgeben könnten. Überstrapazieren Sie diese Art von Parameter jedoch nicht. Diese Maßnahme sollte wirklich eine Ausnahme bleiben, denn sie ist keine ganz saubere Programmierpraxis (Funktionen ändern ihre Parameter normalerweise nicht auf globaler Ebene, und dadurch schleichen sich manchmal hässliche Fehler ein), auch wenn es die Dinge häufig erleichtert und Sie damit clevere Tricks anwenden können.

Ein mögliches Beispiel ist die »automatische« Aktualisierung sogenannter »Laufvariablen«. Laufvariablen sind Variablen, die ihre Werte während einer algorithmischen Schleife ändern. (Ein Sonderfall hiervon sind Zählvariablen in `for()`-Anweisungen.) Ein konkretes Beispiel, das hiervon profitiert, ist der Lauflängen-Kodierungsalgorithmus (RLE) , der weithin bekannt ist, da er in Formaten wie ZSoft's PCX Bildformat verwendet wurde (und noch verwendet wird).

Der RLE-Algorithmus ist ein Komprimierungsalgorithmus, der von der Tatsache profitiert, dass viele Farbbilder mit niedriger Auflösung dieselben Datenbytes wiederholt abspeichern (insbesondere Bitmaps, die nur zwei Farben – schwarz und weiß – enthalten). Sehen wir uns einmal diese Darstellung eines einfachen Rechtecks an:

```
11111111111111111111
10000000000000000001
10000000000000000001
10000000000000000001
10000000000000000001
10000000000000000001
10000000000000000001
10000000000000000001
10000000000000000001
11111111111111111111
```

Mit ein wenig Vorstellungskraft sehen wir ein Rechteck aus Nullen mit einem Rand aus Einsen. Wenn Sie diesen in eine Datei wie »vorliegend« übertragen wollten, benötigten Sie 20 x 10 = 200 Elemente (20 Spalten mal 10 Zeilen). Wir gehen an dieser Stelle einmal davon aus, dass ein Element mindestens einem Byte entspricht und lassen außer acht, dass eine einfache Komprimierung bereits durch Packen dieser Daten in Bits erreicht werden kann.

Was sich hier zeigt, ist, dass die Speicherung der Daten wie vorliegend nicht unbedingt sinnvoll ist. Wenn Sie die Daten jemandem zum Niederschreiben diktieren sollten, würden Sie es nicht als »Eins Eins Eins Eins Eins Eins Eins [...] Eins Null Null Null Null Null [...] Null Eins Eins Eins [...],« vorlesen, sondern würden dem Schreiber »zwanzig Einsen, eine weitere Eins, achtzehn Nullen, [...]« diktieren.

Das Gleiche macht auch der RLE-Algorithmus, der die Anzahl der aufeinanderfolgenden Elemente gleichen Wertes zählt und sie anschließend als »Zähler/Wert«-Paar abspeichert. Nach der Komprimierung könnten die obigen Daten dann folgendermaßen aussehen:

```
21, 1, 18, 0, 2, 1, 18, 0, [...], 18, 0, 21, 1
```

Das erste Element gibt stets den Zähler an, während das zweite Element das Datenelement darstellt. Um dies wieder zu dekomprimieren, müssen Sie lediglich den Zähler lesen und das folgende Datenelement so häufig ausgeben, wie es der Zähler angibt. Mit diesem »Trick« reduzieren Sie die Anzahl der benötigten Elemente von 200 auf 34.

Das Problem bei diesem Algorithmus ist, dass er bei vielen verschiedenen Elementen in einer Zeile mehr Ausgabedaten erzeugt, als ursprünglich Eingabedaten vorhanden waren. Durch Speichern vieler Elemente, die einen Datenzähler von Eins haben, wird dieser Algorithmus damit schnell ineffektiv.

Diese Schwäche lässt sich mit einem kleinen Trick, der sogenannten »Zählerschwelle« umgehen. Jeder Datentyp hat einen spezifischen Zählerbereich: Im obigen Fall überschritt der Zähler beispielsweise niemals 21. Genaugenommen waren die häufigsten Werte 18 und 2. Wenn wir den Zähler nun auf einen Bereich von 1 bis 31 beschränken, können wir auch »reine« Zahlen einfügen, die keine Komprimierung im Eingabestrom benötigen. Der Kompres-

sor spuckt alle Eingabedatenelemente größer als 31 einfach aus, ohne ihnen einen Zähler voranzustellen. Damit erreichen wir für alle Eingabewerte größer als 31 einen 100% optimalen Algorithmus. Alle Werte bis 31 werden zwar immer noch nicht optimal komprimiert, aber dies ist vernachlässigbar.

Um die Daten zu dekodieren, ist es nun wichtig, zwischen reinen Datenelementen und komprimierten Datenelementen zu unterscheiden, was Sie auch in den Listings 2.3, 2.4 und 2.5 sehen können. Hier kommen variable Argumente optimal zum Einsatz.

```
function encode_data()
{

    // do initial setup on our variables
    $current_count = 0;
    $status = read_from_input($current_byte);
    $old_byte = $current_byte;
    $output = array();

    // as long as there's input data, loop
    while($status)
    {
     // check if the current byte matches the last one
     if($old_byte == $current_byte)
     {
         // there's a match, increase counter
         $current_count++;

         // does the counter exceed the threshold?
         if($current_count == COUNTER_THRESHOLD)
         {
          // it does, flush cache and restart
          $output[] = chr($current_count);
          $output[] = $current_byte;

          $current_counter = 0;
         }
     }
     else
     {
             // bytes don't match

             // do we have a cached pair?
             if($current_count > 1)
             {
              // yes we do, write it
              $output[] = chr($current_count);
```

```
              $output[] = $old_byte;

              $current_count = 1;
             }
             else
             {
              // we don't have a cached pair,
              // write literal
              if(ord($old_byte) < COUNTER_THRESHOLD)
             {
                  // this byte could be mistaken as a counter
                  // value, so write a dummy pair
                  $output[] = chr(1);
                  $output[] = $old_byte;
             }
             else
             {
                 // can't be mistaken as counter value, just
                 // write the value directly
                 $output[] = $old_byte;
             }
            }
        }

        // set current byte as old byte
        $old_byte = $current_byte;

        // get new byte and loop
        $status = read_from_input($current_byte);
    }

    return($output);

}
```

Listing 2.3: Der RLE-Komprimierungsalgorithmus

```
function get_encoded_pair(&$count, &$value)
{

    // check input stream
    if(!read_from_input($data_element))
    {
        // no input data available, return
        // zero count and dummy data
        $count = 0;
```

```
        // indicate failure
        return(0);
    }

    // test if this is literal data
    if(ord($data_element) >= COUNTER_THRESHOLD)
    {
        // this is literal data, return
        // count of one and data element
        $count = 1;
        $value = $data_element;
    }
    else
    {
        // this has been a count, assign it
        $count = ord($data_element);

        // try to retrieve the data element
        // itself
        if(!read_from_input($value))
        {
            // input data is corrupted,
            // return zero count
            $count = 0;

            // indicate failure
            return(0);
        }

    }

    // return success
    return(1);

}

function decode_data()
{

    // initialize output array
    $output = array();

    // decompress all data into the array
    while(get_encoded_pair(&$count, &$value))
    {
        for($i = 0; $i < $count; $i++)
```

```
            $output[] = $value;
        }

        return($output);

}
```

Listing 2.4: Der RLE-Expandierungsalgorithmus

```
//
// this declaration must exist and be equal
// both for the compressor and decompressor
//
define("COUNTER_THRESHOLD", 32);

//
// this tool function is needed for both the
// compressor and decompressor to read data
//
function read_from_input(&$data_element)
{
    // This is a dummy function to retrieve a data element
    // from the input data. It could contain code to read
    // from an array, from the standard input, or something
    // completely different.
    // As an example, this function reads from a global
    // file. (Not a good idea to have global files but
    // just for the example's sake.)
    global $file_handle;

    // check if we have reached the end of the file
    if(feof($file_handle))
    {
        // we did, return error
        return(0);
    }

    // we did not encounter the end of the file,
    // so read next element
    $data_element = fgetc($file_handle);

    // return success
    return(1);
}

// include compressor and decompressor
include("compressor.php3");
```

```
include("decompressor.php3");

// define filenames
$original_file = "data.original";
$compressed_file = "data.compressed";
$decompressed_file = "data.decompressed";

// -> all procedures need the global variable $file_handle
// (bad practice but best for a simple example)

// open input file
$file_handle = fopen($original_file, "r");
if(!$file_handle)
    die("Error opening file.");

// encode it
$output = encode_data();

// close input file
fclose($file_handle);

// open output file
$file_handle = fopen($compressed_file, "w");
if(!$file_handle)
    die("Error creating file.");

// write decoded data
for($i = 0; $i < count($output); $i++)
    fputs($file_handle, $output[$i]);

// close output file
fclose($file_handle);

// open input file
$file_handle = fopen($compressed_file, "r");
if(!$file_handle)
    die("Error opening file.");

// decode it
$output = decode_data();

// close input file
fclose($file_handle);

// open output file
$file_handle = fopen($decompressed_file, "w");
```

```
if(!$file_handle)
    die("Error creating file.");

// write decoded data
for($i = 0; $i < count($output); $i++)
    fputs($file_handle, $output[$i]);

// close file
fclose($file_handle);
```

Listing 2.5: Beispiel für die Verwendung des RLE-Prozessors

Dieses Beispiel zeigt sehr schön, wie der Dekodierer und die Leserlogik in kleinere separate Funktionen unterteilt werden können. Der eigentliche Dekodierer ist nur noch wenige Zeilen lang. Die Funktion, welche die Eingabedaten liefert, kann vom Rest des Programmcodes abgetrennt werden, und auch der »Entscheidungsalgorithmus«, der zwischen reinen und kodierten Daten unterscheidet, ist eine separate kleine und leicht zu verstehende Funktion.

Eine clevere Methode ist, reine Daten mit einem Zähler von 1 zurückzugeben, so dass sich die Dekomprimierungsschleife nicht mehr mit ihnen befassen muss. Sie kann die gelieferten Daten einfach unverändert in das Ausgabe-Array schreiben.

Die Fehlerkontrolle könnte zwar noch etwas verbessert werden, aber dies überlassen wir dem Leser. Es ist nicht sonderlich schwer.

Variable Argumentenlisten

Variable Argumentenlisten, die häufig auch als optionale Parameter bezeichnet werden, geben Ihnen die Möglichkeit, Funktionsparameter mit einem Standardwert vorzubelegen. Wenn der Anrufer keinen Wert für das Argument angibt, wird das Standardargument angenommen. Dem Anrufer kann eine Liste mit optionalen Parametern zur Verfügung gestellt werden, die er verwenden kann, aber nicht muss.

Optionale Parameter sind wie folgt definiert:

```
function open_http_connection($hostname, $port = 80, $timeout = 10)
{

    $socket = fsockopen($hostname, $port, $timeout);
    /* rest of the code goes here */

    return($socket);

}
```

```
$regular_socket = open_http_connection("www.myhost.com");
$slow_socket = open_http_connection("www.myhost.com", 80, 20);
$test_socket = open_http_connection("testserver.myhost.com", 8080);
$slow_test_socket = open_http_connection("testserver.myhost.com", 8080,
➥20);
```

Die Funktion `open_http_connection()` akzeptiert ein reguläres Argument namens `$hostname`, das den Namen des Host-Rechners angibt, mit welchem eine Datenverbindung hergestellt werden soll. Zusätzlich hat die Funktion zwei optionale Argumente `$port` und `$timeout`, welche die zu verbindende Portnummer und das Timeout für die Verbindung (in Sekunden) angeben.

Beide haben vorgegebene Standardwerte, die durch das Gleichheitszeichen (=) angegeben werden, gefolgt von dem gewünschten Standardwert. Wenn diese Argumente nicht vom Anrufer ausgefüllt werden, ersetzt PHP die fehlenden Einträge durch die Standardwerte.

Wie Sie an diesen beiden Beispielen sehen, wird im ersten Aufruf nur `$hostname` verwendet. `$port` und `$timeout` fehlen. PHP füllt diese Lücken mit den Standardwerten aus, so dass der Aufruf nun folgendermaßen aussieht:

```
$regular_socket = open_http_connection("www.myhost.com", 80, 10);
```

Sie können trotzdem ihrerseits die optionalen Parameter angeben und die Standardwerte überschreiben, wie der zweite Aufruf zeigt. Der Wert für `$port` ist immer noch 80, aber `$timeout` ist nun auf 20 Sekunden gesetzt.

Sie müssen die Standardargumente angegeben, deren Wert Sie ändern möchten, wie das dritte Beispiel zeigt. Hier ist nur `$port` als 8080 angegeben. `$timeout` fehlt und bleibt daher auf seinem Standardwert von 10 Sekunden.

Die Tatsache, dass PHP keine Möglichkeit hat herauszufinden, welcher Wert zu welchem Parameter gehört, bedeutet, dass Sie alle optionalen Parameter am Ende der Argumentenlisten anfügen müssen. Wenn Sie `$hostname` als einzigen obligatorischen Parameter am Ende der Liste einfügen würden (und `$port` und `$timeout` davor beließen), würde PHP beim ersten Aufruf, bei dem nur `$hostname` angeben ist, vermuten, dass die Zeichenkette für den Namen des Hostrechners der Wert für `$port` wäre, was zu erheblicher Verwirrung führen würde.

Genauso wenig können Sie einfach willkürlich optionale Parameter auswählen, für die Sie Werte liefern möchten. Wenn Sie eine Funktion haben, die drei optionale Parameter hat, und Sie nur den letzten Parameter in der Argumentenliste ändern wollen, müssen Sie trotzdem die Standardwerte für die ersten beiden Parameter angeben. (Dies können Sie auch im zweiten der obigen Beispiele sehen, in dem nur `$timeout` geändert wird.)

PHP 4.0 kann Argumente wirklich variabel verwalten. Eine Funktion kann mehr Argumente haben als die Funktionsdefinitionslisten, und Sie können mit den Funktionen func_get_args(), func_num_args() und func_get_arg() auf eine beliebige Anzahl von Argumenten zugreifen.

Der Rückgabewert von func_get_args() ist ein indiziertes Array, das von links nach rechts mit allen Argumentwerten aufgefüllt wird, die an die Funktion übergeben wurden:

```
function show_arguments()
{

    $argument_array = func_get_args();

    for($i=0; $i<count($argument_array); $i++)
    {
        print("$i => $argument_array[$i]<br>");
    }

}

show_arguments("Leftmost", "Middle", "Rightmost");
```

Die Funktion func_num_args() gibt die Anzahl der übergebenen Argumente zurück. func_get_arg() gibt ein spezifisches Argument zurück. func_get_arg(0) würde beispielsweise das erste Argument zurückgeben.

Variable Variablennamen

Variable Variablennamen. Für jene, die dieses Konzept noch nicht kannten (und für uns, als wir es zum ersten Mal hörten) – einfach verrückt! Variable Variablennamen dienen dazu, auf Variablen zuzugreifen, deren Namen Sie vorher noch nicht kennen und die Sie erst während der Laufzeit erstellen. Diese Funktion ist durch den Interpretationscharakter von PHP möglich. PHP wandert einfach den Programmcode ab und übersetzt alles, was es findet, in etwas möglichst Sinnvolles. Der folgende Programmcode zeigt ein einfaches Beispiel für variable Variablennamen:

```
<?

$my_var = "hello";

$$my_var = 1;

?>
```

In der zweiten Zeile wird `$my_var` das Präfix `$$` vorangestellt. Dies ist das Grundprinzip von variablen Variablennamen. Natürlich könnten Sie variable Variablennamen verschachteln, um variable variable Variablennamen zu erhalten usw. Aber auch mit variablen Variablennamen können Sie einige nette Sachen anstellen.

Ein Beispiel aus dem Leben: phpPolls, das Stimmabgabeprogramm, das wir in Kapitel 1 »Entwicklungskonzepte« beschrieben haben, verwendet variable Variablennamen. Um zu verhindern, dass Wähler in derselben Wahl mehrfach abstimmen, basiert einer der Schutzmechanismen auf Cookies. Sobald ein Wähler wählt, wird ein Cookie gesetzt und mit einem Namen versehen, der einen konfigurierbares Präfix und eine eindeutige ID erhält, welche die Wahl kennzeichnet. Da Cookies wieder in den globalen Namensraum zurückgeführt werden, prüft phpPolls bei der Stimmabgabe eines Benutzers, ob es eine globale Variable mit dem Namen des Cookies gibt, den es vorher erstellt hatte. Wenn dies der Fall ist, nimmt phpPolls die Stimmabgabe nicht an.

Für diese Aufgabe sind variable Variablennamen sehr gut geeignet. Hier ein Auszug aus dem phpPolls-Quellcode:

```
$poll_object = mysql_fetch_object($poll_result);
$poll_timeStamp = $poll_object->timeStamp;

$poll_cookieName = $poll_cookiePrefix.$poll_timeStamp;

// check if cookie exists
if(isset($$poll_cookieName))
{
    // cookie exists, invalidate this vote
    $poll_voteValid = 0;
}
else
{
    // cookie does not exist yet, set one now
    setCookie("$poll_cookieName", "1", $poll_cookieExpiration);
}
```

Zunächst wird die eindeutige ID für den Cookie gelesen, die aus dem Zeitstempel für die Wahl besteht, und dann mit dem Präfix des Cookies `$poll_cookiePrefix` zu dem Namen der gewünschten Variablen `$poll_cookieName` zusammengesetzt. Mithilfe von `isset()` wird das Vorhandensein der Variable (und damit des Cookies) überprüft und eine entsprechende Aktion eingeleitet.

Variable Funktionsnamen

Was wir über variable Variablennamen gesagt haben, gilt auch für Funktionsnamen. Auch Funktionsnamen können mithilfe von Variablen erzeugt werden, was eine dynamische Datenverarbeitung, Installation von modifizierbaren Rückrufen u.ä. ermöglicht. Anstatt Funktionsnamen fest im Programm einzugeben, können Sie Zeichenkettenvariablen verwenden, um die Funktion anzugeben, die Sie aufrufen möchten:

```
function my_func($a, $b)
{

    print("$a, $b");

}

$function = "my_func";

$function(1, 2);
```

Nach der Deklaration von `my_func()` wird die Variable `$function` der Zeichenkette `my_func` zugeordnet. Da diese Zeichenkette mit dem Namen der Funktion identisch ist, die Sie aufrufen möchten, können Sie diese beim Aufruf von `my_func()` verwenden.

Dies ist natürlich ein recht einfaches Beispiel für variable Funktionsnamen. Sie sind sehr nützlich, wenn Sie, je nach Anzahl der variablen Flags, zwischen mehreren Funktionen hin- und herschalten müssen.

Angenommen, Sie wollten Email-Anhänge dekodieren. Diese können verschieden Formate haben, z.B. base64 oder uuencoded, um nur zwei Beispiele zu nennen. Wenn Sie einen »geschlossenen« Parser generieren, der nur eine der beiden Kodierungen erkennt, lässt sich dieser nur schwer erweitern. Sobald Sie neue Formate benötigen, stecken Sie fest. Dies ist ein klassischer Fall für variable Funktionsnamen:

```
function decode_base64($encoded_data)
{

    // do something with the encoded data

    return($decoded_data);

}

function decode_uuencoded($encoded_data)
{
```

```
    // do something with the encoded data

    return($decoded_data);

}

$mail_text = fetch_mail();
$encoder_type = determine_encoding($mail_text); // returns: "base64" for

➥Base64
                                                 // returns: "uuencoded" for
                                                    ➥UUEncoded
$decoder = "decode_".$encoder_type;

$decoded_data = $decoder($mail_text);
```

Dieser Programmcode stellt automatisch die korrekte Bearbeitungsroutine für die Eingabedaten fest. `determine_encoding()` gibt die Zeichenfolge zurück, die den zu entschlüsselnden Datentyp angibt, für den eine entsprechende Funktion vorhanden sein muss. Der Name der aufzurufenden Funktion wird dann in `$decoder` geschrieben und sofort aufgerufen.

Der Nachteil dieses Verfahrens ist, dass es nicht sehr sauber ist. Sie werden kaum ein »Standard« verhalten feststellen. Das Entschlüsselungsverfahren ist vollständig dynamisch und könnte zusammenbrechen, wenn `determine_encoding()` ein bedeutungsloses Ergebnis erzeugt. Es ist jedoch eine recht komfortable Art und Weise Eingabedaten zu verarbeiten. Sobald neue Kodierungstypen erscheinen, brauchen Sie nur eine Funktion mit passendem Namen zu erzeugen und `determine_encoding()` anzupassen, um die entsprechende Zeichenfolge zu erhalten.

Solange Sie `determine_encoding()` absturzsicher gestalten, so dass es eine sinnvolle Zeichenfolge zurückgibt (selbst wenn es nur ein Dummy ist), sind wir mit dieser Methode vollkommen einverstanden. Solange Sie sicherstellen können, dass Ihr Skript sich während der gesamten Laufzeit in einem definierten Status befindet, ist nichts dagegen einzuwenden, diese dynamische Datenverwaltung für Produktionsumgebungen einzusetzen.

Ein realistisches Beispiel eines Skripts, das extensiven Gebrauch von variablen Funktionsnamen macht, ist phpIRC (das wir im folgenden Kapitel besprechen werden). phpIRC ist eine IRC-Schicht für PHP, die über eine benutzerfreundliche API einen Zugriff auf IRC(Internet Relay Chat)-Netzwerke bietet. Da die Verwaltung der Eingabedaten nicht-linear ist und vollständig vom Benutzer abhängt, verwendet phpIRC für jedes ankommende Paket eine Reihe von Ereignisklassifizierungen. Der Benutzer kann für jedes Ereignis in phpIRC Bearbeitungsroutinen installieren, die auf jede Art von ankommenden Daten angemessen reagieren. phpIRC speichert die Namen der Rück-

ruffunktionen in einem internen Array. Sobald Daten ankommen, durchsucht es sein Rückruf-Array nach Funktionen, die auf diesen entdeckten Datentyp passen und ruft alle passenden Funktionen auf einmal auf. Dies ermöglicht, ähnlich wie bei dem vorangegangenen Email-Beispiel, eine dynamische Datenverarbeitung, was sich bezahlt macht, wenn Sie nur zur Laufzeit entscheiden können oder wollen, wie Sie mit ankommenden Ereignissen verfahren.

Wenn Sie diesen Ansatz weiter verfolgen, können Sie variable Funktionsnamen einsetzen, um das Verhalten Ihrer Skripte zur Laufzeit zu ändern und gleichzeitig benutzerdefinierte Plugins installieren, welche zur Laufzeit automatisch an den Programmcode angehängt werden Damit erhalten Sie zusätzliche Funktionen für das Skript, ohne dass Sie dafür eine einzige Programmzeile ändern müssen.

2.7 Polymorphisierung und selbstmodifizierender Code

Die Nachteile von variablen Funktionsnamen (und teilweise von variablen Variablennamen) ist, dass Sie stets einen »festen« Programmteil haben müssen (im Fall der variablen Funktionsnamen eine Liste mit zuvor deklarierten Funktionen, die Sie verwenden können) und einen »variablen« Teil (der Teil, der die Funktionsnamen in eine Variable umwandelt und dann die Funktion aufruft, dessen Namen erstellt wurde). Dies bedeutet, dass Sie für jeden möglichen erstellbaren Namen vorher eine Funktion generieren müssen, damit das Programm korrekt funktioniert – was eine Form von Beschränkung darstellt.

Dies können Sie durch vollständig dynamische Programme umgehen, Programme die sich selbst »en passant« generieren. Diese Idee stammt ursprünglich aus den »frühen« Tagen der Programmierung und wurde z. T. von Spieleprogrammierern und Virusschreibern erfunden.

Es begann alles mit sich selbst modifizierendem Programmcode. Bei den inneren Schleifen von Spielen, beispielsweise den Prozeduren, die zum Kopieren des Pufferinhalts in den Bildschirmspeicher dienten, war und ist Zeit der kritische Faktor. Da die Verarbeitungszeit jedoch keine unerschöpfliche Quelle war und ist, mussten sich die Entwickler etwas einfallen lassen, um das Beste aus ihrem Werk herauszuholen.

Häufig mussten in der innersten Schleife eine Reihe von Entscheidungen getroffen werden. Wenn beispielsweise die Kopierroutine des Puffers bei bestimmten Ereignissen nur jede zweite Zeile in den Bildschirmspeicher ausgeben sollte, mussten Sie ein paar `if()/then`-Konstrukte in einen Programm-

teil einbetten, der diese am wenigsten benötigte. Diese Konstrukte kosteten wertvolle Verarbeitungszeit, und da diese innerste Routine etwa 80% der gesamten Verarbeitungszeit ausmachte, bedeute eine Halbierung dieser Verarbeitungszeit durch Entfernen dieser Konstrukt eine 40%ige Steigerung der verfügbaren Rechenkapazität.

Für jeden Fall eine Reihe von Routinen zu schreiben, wäre auch nicht sehr hilfreich gewesen. Sie hätten trotzdem Speicherplatz verschwendet und die Entscheidungen nur an eine andere Stelle verschoben. Sie können damit zwar die Gesamtleistung verbessern, aber eine optimale Leistung erreichen Sie nicht.

Deshalb wurde die Technik des selbstmodifizierenden Programmcodes eingeführt. Wo ein Programmteil früher eine der Bedingungen der innersten `if()/then`-Schleife geändert hätte, indem er die zuständigen Flags entsprechen gesetzt hätte, programmierte er die innere Schleife einfach neu, und zwar so, dass sie wie gewünscht agiert, d.h. so als hätte sie die Flags ausgewertet. Die Modifikationen, die hierfür notwendig waren, betrafen häufig nur das Ändern von ein oder zwei Bytes und waren meist nicht aufwendiger als das Setzen von Flags. Dies funktionierte natürlich nur auf Maschinencode-Ebene und war extrem systemabhängig – dafür aber auch sehr mächtig.

Die Entwickler von Virusprogrammen haben dieses Verfahren schließlich auf die Spitze getrieben, indem sie polymorphe Programme entwickelten. Polymorphe Programme heißen polymorph, weil sie ihren eigenen Programmcode ändern, während sie noch dieselbe Aufgabe ausführen. Die einfachste Methode, um einen polymorphen Programmcode zu entwickeln, bestand darin, die Viren zu komprimieren und jedes Mal einen anderen Komprimierungsalgorithmus oder andere Komprimierungsparameter auszuwählen. Nach jeder Komprimierung wurde so der Bytecode geändert, doch nach der Dekomprimierung wurde das ursprüngliche Programm wiederhergestellt. (Versuchen Sie einmal die ZIP-Archive anzusehen, die dieselben Daten auf verschiedenen Komprimierungsebenen enthalten: Sie sehen ganz verschieden aus, haben aber alle nach der Expansion wieder dieselben Daten.) Der kompliziertere Ansatz war, Anweisungsblöcke dynamisch neu zu gruppieren und gleichzeitig die algorithmische Struktur beizubehalten. Dieses Verfahren erforderte manchmal sehr anspruchsvollen Programmcode, war aber auch sehr effektiv. Da jede Methode dazu führte, dass der Bytecode geändert wurde, wurden verschiedene Signaturen für die einzelnen Infektionen desselben Virus erzeugt, so dass es für Antivirus-Programme nahezu unmöglich war, sie zu entdecken, während die Viren andere Programme lustig weiter infizieren konnten.

Was hat dies mit PHP zu tun? Natürlich können Sie keine solchen polymorphen Programme erstellen. Die PHP-Architektur verhindert während der Laufzeit Änderungen des Programmcodes, der bereits analysiert ist. Trotz-

dem enthalten sie nützliche Hinweise. Eine Möglichkeit sind dynamische Funktionsparser, auf die wir im nächsten Abschnitt eingehen.

2.7.1 Dynamischer Funktionsgenerator

Als wir an diesem Buch schrieben, fragte jemand aus der deutschen PHP-Mailingliste, ob wir eine Möglichkeit wüssten, vom Benutzer eingegebene mathematische Funktionen zu verwalten. Er wollte wissen, wie er eine vom Benutzer eingegebene Funktion mithilfe einer Webformel in PHP grafisch darstellen kann. Er wusste nicht, wie er mit dem Text verfahren sollte. Wie lässt sich etwas wie $f(x) = m * x + b$ in ein Diagramm übertragen?

Die Diskussion führte schnell in eine traditionelle Richtung. Jeder fing an, im großen, im wirklich großen Rahmen zu denken, anstatt die einfachen, offensichtlichen Dinge in Betracht zu ziehen. Uni sono (aus dem Lateinischen: mit einer Stimme) lautete der allgemeine Vorschlag zur Lösung des Problem folgendermaßen:

1. Analysiere die Eingabedaten.
2. Generiere eine syntaktisch analysierte Darstellung (etwas in Bezug auf Compilertechniken).
3. Lasse einen Prozessor über die analysierte Darstellung laufen und erzeuge Schritt für Schritt numerische Ausgabedaten.

Für unser konkretes Beispiel $f(x) = m * x + b$ sähe dies folgendermaßen aus:

1. Sie stellen fest, dass es sich um eine Funktion von x handelt, in der m und b Variablen sind.
2. Sie erzeugen zur einfacheren Handhabung eine Struktur, um den Text der Funktion intern darzustellen, z.B. in einer Baumstruktur, welche die Abhängigkeiten der Variablen vom Multiplikationszeichen (*) bzw. Additionszeichen (+) angibt.
3. Sie setzen für x (der Variablen, von der die Funktion abhängt) verschiedene Werte ein, ermitteln y, speichern das Ergebnis und interpolieren die Daten.

Dies ist der in den Universitäten gelehrte Ansatz, den wir bereits in komplexen Quellcodes u.ä. gesehen haben. Niemand scheint sich von gewohnten Lösungsansätzen frei machen und eine innovativere Lösung suchen zu können. Haben Sie schon einmal darüber nachgedacht, was PHP tut, wenn es ihre Skripte interpretiert?

1. Es analysiert die eingegebenen Quelldaten.
2. Es erzeugt eine analysierte Darstellung.

3. Es lässt einen Prozessor über die analysierte Darstellung laufen.

Gut, die ist etwas vereinfacht, aber im Grunde genommen ist es das, was wir brauchen. Warum sollten wir also die Eingabefunktion nicht in einen gültigen PHP-Code umwandeln und dann PHP die Arbeit für uns machen lassen? Wie Sie bereits in diesem Kapitel gesehen haben, unterstützt PHP dynamisches Programmieren, so dass sich die gesamte Aufgabe möglicherweise recht einfach bewältigen lässt.

Tatsächlich sind die regulären Ausdrücke, die zur Umwandlung einer einfachen mathematischen Funktion in PHP-Code benötigt werden, extrem einfach. Wenn wir davon ausgehen, dass alle Variablen aus einem einzelnen Zeichen bestehen und nur in PHP zulässige mathematische Operatoren verwendet werden (+, –, *, usw.), lässt sich die Aufgabe in einer einzigen Zeile realisieren:

```
$php_code = ereg_replace("([a-zA-Z])", "$\\1", $input_function);
```

Diese Zeile wandelt *m * x + b* in *$m * $x + $b* um. Wenn wir diesen regulären Ausdruck in ein wenig Programmcode einbetten und ein paar vereinfachende Annahmen machen, können wir sehr schnell einen dynamischen Funktionsplotter erstellen, wie Listing 2.6 zeigt.

```
//
// define global constants
//
define("PLOT_MIN", 0.1);
define("PLOT_MAX", 100);
define("PLOT_STEP", 0.5);
define("DIAGRAM_HEIGHT", 300);
define("DIAGRAM_HORIZON", 150);

function parse_function($in_string)
{

    // define a custom function header
    $header = "";
    $header = $header."function calculate(\$req_code, \$x)\n";
    $header = $header."{\n";
    $header = $header."eval(\$req_code);\n";

    // define a custom function footer
    $footer = "\n}\n";

    // convert all characters to PHP variables
    $out_string = ereg_replace("([a-zA-Z])", "$\\1", $in_string);
```

```
    // prepend header, create equation, and append footer
    $out_string = $header."return(".$out_string.");\n".$footer;

    // return result
    return($out_string);

}

function create_image()
{
    // export this variable
    global $color_plot;

    // we calculate the X scale based on the plot parameters
    // the diagram height is fixed as we do not check for the
    // function's extreme points
    $width = PLOT_MAX / PLOT_STEP;
    $height = DIAGRAM_HEIGHT;

    $image = imagecreate($width, $height);

    // allocate colors
    $color_backgr = imagecolorallocate($image, 255, 255, 255);
    $color_grid = imagecolorallocate($image, 0, 0, 0);
    $color_plot = imagecolorallocate($image, 255, 0, 0);

    // clear image
    imagefilledrectangle($image, 0, 0, $width - 1, $height - 1, $color_
    ➥backgr);

    // draw axes
    imageline($image, 0, 0, 0, $height - 1, $color_grid);
    imageline($image, 0, DIAGRAM_HORIZON, $width - 1, DIAGRAM_HORIZON,
    ➥$color_grid);

    // print some text
    imagestring($image, 3, 10, DIAGRAM_HORIZON + 10, PLOT_MIN, $color_
              ➥grid);
    imagestring($image, 3, $width - 30, DIAGRAM_HORIZON + 10, PLOT_MAX,
    ➥$color_grid);

    // return image
    return($image);

}
```

```
function plot($image, $x, $y)
{
    // import the color handle
    global $color_plot;
    // set these as static to "remember" the last coordinates
    static $old_x = PLOT_MIN;
    static $old_y = 0;

    // only plot from the second time on
    if($old_x != PLOT_MIN)
        imageline($image, $old_x / PLOT_STEP, DIAGRAM_HEIGHT -
        ➥($old_y + DIAGRAM_HORIZON), $x / PLOT_STEP, DIAGRAM_HEIGHT -
        ➥($y + DIAGRAM_HORIZON), $color_plot);

    $old_x = $x;
    $old_y = $y;

}

// see if we've been invoked with a function string set
if(!isset($function_string))
{
    // no, there's no function string present,
    // generate an input form
    print("<html><body>");
    print("<form action=\"".basename($PHP_SELF)."\" method=\"post\">");
    print("Function definition: <input type=\"text\"_
          ➥name=\"function_string\" value=\"(m*x+b)/(x/3)\"><br>");
    print("Required PHP code: <input type=\"text\" name=\"req_code\"
          ➥value=\"\$m = 10; \$b = 20;\"><br>");
    print("<input type=\"submit\" value=\"Parse\">");
    print("</form>");
    print("</body></html>");
}
else
{
    // translate input function to PHP code
    $parsed_function = parse_function($function_string);

    // *** NOTE: security holes! (see book contents) ***
    eval($parsed_function);

    // create image
    $image = create_image();
```

```
    // plot the function
    for($x = PLOT_MIN; $x < PLOT_MAX; $x += PLOT_STEP)
    {
        $y = calculate($req_code, $x);
        plot($image, $x, $y);
    }

    // set content type
//  header("Content-type: image/gif");
    header("Content-type: image/png");

    // send image
//  imagegif($image);
    imagepng($image);

}
```

Listing 2.6: Dynamische Funktionsparser und -plotter

Das Skript ist ausführbar. Sie können es direkt in Ihrem Browser verwenden. Beim ersten Aufruf wird es feststellen, dass Sie noch keine Funktion zum Plotten bereitgestellt haben, und ein kleines Eingabeformular anzeigen, wie Abbildung 2.13 zeigt.

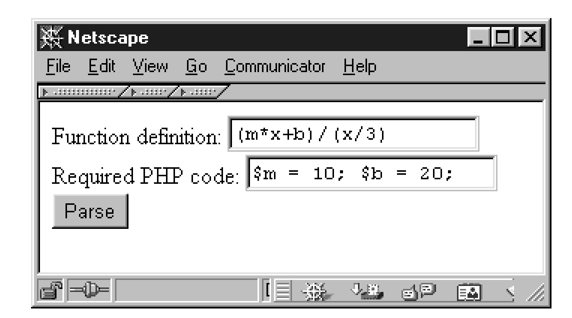

Abbildung 2.13: Eingabeformular des dynamischen Funktionplotters

Warnung: Die hier verwendete Technik, mit welcher der PHP-Code direkt mit `eval()` ausgeführt werden kann, sollte niemals (wir wiederholen: niemals!) in dieser Form in Produktionsskripten verwendet werden. Die Ausführung von Benutzerprogrammen reißt ein riesiges Sicherheitsloch in Ihre Programme. Damit könnte jeder etwas wie `system("rm -r /*")` senden, womit alle Daten gelöscht würden, auf die Ihr Webserver Zugriff hat. Wir haben es hier gemacht, weil wir uns auf die dynamische Erstellung und Ausführung von Programmcode konzentrieren wollten. Eine ausführliche Diskussion über die Möglichkeit, Ihre Skripte zu sichern (und die Ausführung von böswilligem Programmcode zu vermeiden), finden Sie in Kapitel 4 »Webanwendungskonzepte« und Kapitel 5 »Grundlegende Webanwendungsstrategien«.

Das erste Feld gibt die Funktion an, die grafisch dargestellt werden soll. Dieses Beispiel macht die Annahme, dass x die einzige Variable ist, von der die Funktion abhängt. Im zweiten Feld können Sie PHP-Code eingeben, der vor der Auswertung der Funktionsanweisung ausgeführt wird, damit Zuordnungen zu Konstanten gemacht werden können (in unserem Fall *m* und *b*).

Zum jetzigen Zeitpunkt klicken Sie einfach auf »Parse«. Abbildung 2.14 zeigt was als Nächstes erscheint.

Wie kam das Skript jetzt von dem Eingabeformular zu dieser grafischen Ausgabe? Sehen wir uns die inneren Vorgänge Schritt für Schritt an.

Nachdem Sie das Eingabeformular abgeschickt haben, beginnt das Skript mit der Ausführung der `else()`-Klausel der wichtigsten `if()`-Anweisung. Zunächst wird folgende Funktion aufgerufen:

```
// translate input function to PHP code
$parsed_function = parse_function($function_string);
```

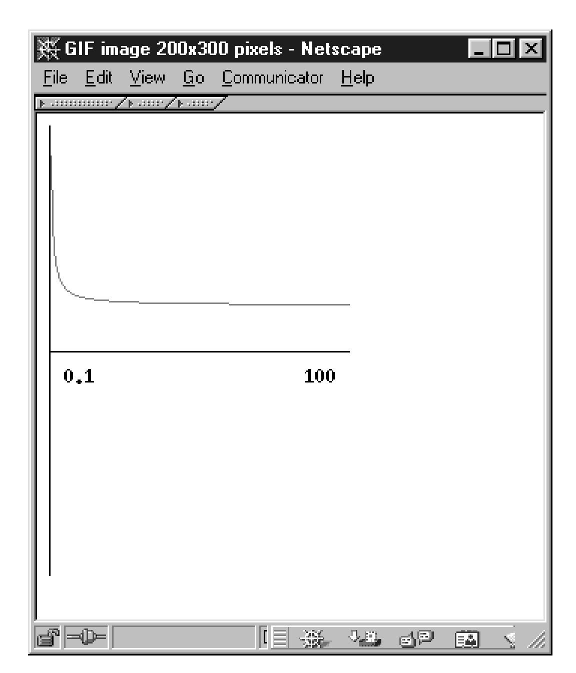

Abbildung 2.14: Beispielausgabe des Funktionplotters

`parse_function()` erzeugt den PHP-Code aus den vom Benutzer gelieferten Daten, indem es auf diese einen regulären Ausdruck anwendet. Um die mathematische Funktion einfacher nutzen zu können, wurde sie in eine kleine Funktion eingebettet, welche die entsprechenden Werte den Konstanten zuordnet (durch erneute Referenz auf die Benutzereingaben). Anschließend wird die mathematische Anweisung ausgeführt und das Ergebnis an den Anrufer zurückgesendet.

Die von `parse_function()` generierte Funktion sieht für unser Beispiel $(m * x + b) / (x / 3)$ folgendermaßen aus:

```
function calculate($req_code, $x)
{
    eval($req_code);
    return(($m * $x + $b) / ($x / 3));
}
```

`$req_code` enthält die Eingabe aus dem zweiten Formularfeld, in diesem Beispiel *$m = 10; $b = 20;*. Wenn wir dies mit `eval()` ausführen, erhalten wir die korrekte Variablenzuordnung für die nächste Zeile, welche bereits die gesamte Berechnung vornimmt. Und das war es!

Hinweis: Wichtige Informationen über die Anweisung `eval()` finden Sie in der vorangegangenen Warnung!

Jetzt wird die Funktion nur noch grafisch dargestellt. Die `for()`-Schleife wird für einen vordefinierten Bereich durchlaufen, und mittels `calculate()` werden bei jedem Durchgang die Y-Werte für die Kurve ermittelt.

2.7.2 Selbstmodifizierender Zähler

Ein einfaches Beispiel für selbstmodifizierende Zähler sind Trefferzähler. Normalerweise würden Trefferzähler von Protokolldateien berechnet oder aus der Datenbank ausgelesen. Eine einfachere Methode ist jedoch die Verwendung von »eingebauten« Zählern. Eingebaut bedeutet, dass sich der Programmcode und die Daten für den Zähler in derselben Datei befinden:

```
$counter = 0;
//////////////////////////////////
// Do not modify above this point
//////////////////////////////////

// increase counter
$counter++;

// write counter back to ourselves
$file = fopen(basename($PHP_SELF), "r+");
fputs($file, "<?\n\$counter = $counter;");

// print counter (or do something else with it)
print("$counter hits so far");
```

In der ersten Zeile wird der Zähler auf Null zurückgesetzt. Die nächste Zeile setzt ihn hoch, und nun kommt der interessante Teil: Der Programmcode öffnet seine eigene Datei und ersetzt die erste Zeile. Dies führt dazu, dass die Datei anders interpretiert wird, wenn sie von PHP das nächste Mal verarbeitet wird. Die Quelle sähe folgendermaßen aus:

```
$counter = 1;
//////////////////////////////////
// Do not modify above this point
//////////////////////////////////

// increase counter
$counter++;

// write counter back to ourselves
```

```
$file = fopen(basename($PHP_SELF), "r+");
fputs($file, "<?\n\$counter = $counter;");

// print counter (or do something else with it)
print("$counter hits so far");
```

Jetzt setzt die erste Zeile `$counter` auf 1 und nicht auf 0. Bei jedem weiteren Durchlauf durch die Datei wird sich die erste Zeile ändern und die Anzahl der Treffer angeben, die es für die Datei bisher gab.

Hinweis: Gleichzeitige Zugriffe mehrerer Anrufer kann dieses Programm nur schwer bewältigen. Es könnte vorkommen, dass zwei PHP-Prozesse die Datei zur selben Zeit lesen (und gleichzeitig in sie schreiben), was zu einem falschen Trefferzähler führen würde. In einer Umgebung mit umfangreichem Datenverkehr sollten Sie diese Technik nur mit entsprechenden Verarbeitungssperren einsetzen.

2.8 Zusammenfassung

In diesem Kapitel haben Sie eine Menge über die erweiterte Syntax von PHP und gute Programmierpraktiken erfahren. Wir haben Ihnen gezeigt, wie Sie Konstanten mit `define()` erzeugen. Anschließend sind wir auf die kniffligeren Aspekte von Arrays eingegangen und haben Ihnen erklärt, dass Sie `list()/each()` zur Abarbeitung von Hash-Codes verwenden sollten. Wir erläuterten die Funktionen der objektorientierten Programmierung in PHP, zeigten, wie und wann Sie diese verwenden, und sagten Ihnen, wann Sie besser bei der prozeduralen Programmierung bleiben sollten. Da PHP eine interpretierte Sprache ist, lässt sie viele Funktionen zu, die sich in konventionell kompilierten Programmiersprachen nur schwer implementieren ließen: variable Variablen und Funktionen, selbstmodifizierender Code und Laufzeitauswertung von Quellcode. Mit diesem Wissen sind Sie gut gerüstet für die erweiterte PHP-Programmierung und einen großen Schritt weiter auf Ihrem Weg zum PHP-Experten.

3 Programmentwicklung: ein Praxisbeispiel

Vermeide Ärger, bevor er entsteht.
Bringe die Dinge in Ordnung, bevor sie existieren.
Die große Pinie entspringt aus einem kleinen Spross.
Eine Reise von tausend Meilen
beginnt mit einem einzigen Schritt

Programmentwicklung ist ein so breites Feld, dass es auch in einem einzigen Buch nicht vollständig abgehandelt werden kann. Der Begriff »Programmentwicklung« umfasst praktisch alle Einzelschritte der Entwicklung von der Konzeption der Datenstruktur, Flussdiagramme und Diagramme über die Beziehung der Einheiten bis zum Entwurf des Programmcodes, der Dokumentation und allem dazwischen. Da es jedoch ein sehr wichtiges Thema ist, wollten wir es nicht ganz aus diesem Buch streichen, sondern erörtern die Frage der Programmentwicklung gezielt an einem praktischen Beispiel, nämlich phpChat. In diesem Kapitel erhalten Sie, ähnlich einer erweiterten Programmfallstudie, einen tiefergehenden Einblick in dieses Echtzeit-Chatserver-Programm, das mit PHP realisiert wurde. Wir hoffen, dass Sie aus diesem nützliche Informationen und Verfahren für sich entdecken, die Sie bei Ihrer nächsten Programmentwicklung einsetzen können.

Viele der umrandeten Texte in diesem Kapitel enthalten Anmerkungen über übliche Techniken zur Programmentwicklung, die Sie direkt an dem vorgeschlagenen Beispiel ausprobieren können (oder allgemein an phpChat) und sich für Ihr nächstes Projekt merken sollten.

> **Hinweis:** Eine mehr theoretische und kürzere Erörterung zum Thema Programmentwicklung finden Sie in Kapitel 7 »Anwendungen der Spitzentechnologie«.

3.1 Projektüberblick

Wenn Sie ein Programm entwickeln, beginnen Sie damit, sich zu überlegen, was die Anwendung tun soll. Im Falle von phpChat soll das Programm einen browser-basierten Chat-Service bereitstellen.

Dieser Chat sollte folgende Funktionen umfassen:

- *Echtzeit-Chat*: keine verzögerte Weiterleitung von Nachrichten und kein Auffrischen

- *Keine clientseitige Programmierung*: Der Browser sollte nur reines HTML (und möglicherweise JavaScript) als Eingabe erhalten.
- *Netzwerkfähig*: Es sollte möglich sein, Chat-Boxen zu verknüpfen.
- Generisch: Machen Sie so wenig Annahmen wie möglich über die Zielsysteme und führen Sie wenig Anforderungen wir möglich ein.
- *Kein Konzeptzwang*: Trennen Sie Programmcode und Seitenlayout.
- *Leicht* zu verwenden und zu verwalten
- *Unbegrenzt*e Anzahl von Clients und Chat-Räumen

Wenn Sie soweit gekommen sind und wissen, was Ihr Programm tun soll, müssen Sie das Konzept auswerten und einen detaillierten Plan ausarbeiten und festlegen, wie das Programm strukturiert werden soll.

Nehmen Sie sich die Zeit, alle Anforderungen niederzuschreiben. Es ist sehr hilfreich und später eine gute Gedächtnisstütze.

Wenn Sie das Programm zusammen mit einem Kunden entwerfen, wird dieser Schritt als »Erstellen einer Spezifikation« bezeichnet. An diesem Punkt kann der Kunde das Layout des Programms noch beeinflussen. Dies ist sehr wichtig, da das Programm den in diesem Schritt angegebenen Anforderungen entsprechen muss, denn ansonsten wird es vom Kunden nicht abgenommen.

Der Kunde hat immer recht – auch wenn er unrecht hat

Kunden, die eine Programmentwicklung bei Ihnen in Auftrag geben, haben häufig nicht die Sachkenntnis, ein solches Programm selbst zu entwickeln. Deshalb heuern sie Sie an. Wenn Sie die Anforderungen mit Ihrem Kunden besprechen, leiten Sie ihn ein wenig. Wenn der Kunde beispielsweise sagt: »Ich möchte ein Chat-Programm, das ganzseitige Bilder jedes Chatters anzeigt und mindestens einmal pro Sekunde aufgefrischt wird« könnten Sie einen Gegenvorschlag machen: »Wäre es nicht besser, neben jede Zeile Miniaturansichten zu platzieren? Die meisten ihrer Chatter werden nicht genügend Bandbreite haben, um ganzseitige Bilder anzuzeigen.«

Seien Sie aber vorsichtig. Bestehen Sie niemals auf Ihrem Standpunkt, es sein denn, Ihr Kunde möchte nicht realisierbare Funktionen haben. Immerhin bezahlt der Kunde Sie, damit Sie seine Vorstellung verwirklichen. Um Ihren Vertrag nicht zu gefährden, müssen Sie möglicherweise vorübergehend eine schlechte Lösung akzeptieren (wenn Sie feststellen, dass Sie den Kunden nicht überzeugen können, den besseren Weg zu wählen) und diese später ändern, wenn der Kunde einsieht, dass seine Strategie nicht funktioniert.

Für dieses Projekt übernehmen Sie die Rolle des Projektleiters und wir Autoren werden Ihre Kunden sein. Da wir nette Kunden sind, werden wir nicht darauf bestehen, dass Sie die Details dieses Programms schriftlich fixieren. Wir überlassen es Ihnen, den Rest zu entwerfen. Wann immer in diesem Kapitel eine Wahl

oder Entscheidung getroffen werden kann, versuchen Sie ihre eigenen Entscheidungen zu treffen. Werten Sie alle Fakten sorgfältig aus und vergleichen Sie ihre Ergebnisse mit den Schlussfolgerungen, die wir im Buch getroffen haben.

3.2 Vergleich von Technologien

Bevor Sie über das Layout des Programmcodes nachdenken, gibt eine Phase, die wir hier spontan als »Dinge zusammenbringen« bezeichnen. Dies ist ein Zwischenschritt zwischen der Idee und dem Erstellen der Spezifikation bzw. des Codelayouts. Es geht darum, die innere Funktionsweise und deren Basis herauszufinden.

Um dies zu verdeutlichen, lassen Sie uns zum Beginn zurückkehren:

- Was wollen wir erstellen?
- Wie wollen wir es erstellen?
- Gibt es bereits Realisierungen hierfür?
- Gibt es ein ähnliches System, das fast dieselbe Aufgabe erfüllt?
- Wenn ja, können wir etwas von diesem Design wiederverwenden?
- Können wir fremde Techniken verwenden, z.B. um unser System zu erweitern?

Fragen über Fragen.

Die erste ist einfach zu beantworten. Wir möchten ein Chat-System erzeugen. Wie? Nun, mit PHP und irgendwie auf der Server-Seite. Mehr wissen wir zu diesem Zeitpunkt noch nicht.

Gibt es bereits irgendwelche Chat-Systeme oder ähnliches? Ja, die gibt es. Da fängt bereits auf Ihrer Shell an: Der Befehl »talk« ermöglicht Ihnen, mit anderen Benutzern zu kommunizieren, die Sie über eine gültige Netzwerkverbindung (oder über eine lokale Verbindung) erreichen, wie Abbildung 3.1 zeigt.

Er ist zwar nicht so leistungsfähig, wie wir es gerne hätten, aber immerhin ein Anfang. Als Nächstes könnten wir im Web surfen, um Seiten zu suchen, die eine der (heute schon fast obligatorischen) Chat-Verbindungen haben. Sie unterscheiden sich zwar stark im Aufbau und Layout, aber die meisten lassen sich auf die folgenden Punkte reduzieren:

- Java für aufwendige Schnittstellen, auch wenn einige reines HTML verwenden
- Ein proprietäres Protokoll mit einem einzelnen Server (oder einfach datenbankbasiert)
- Kaum vordefinierte Räume
- Kaum vordefinierte Befehle

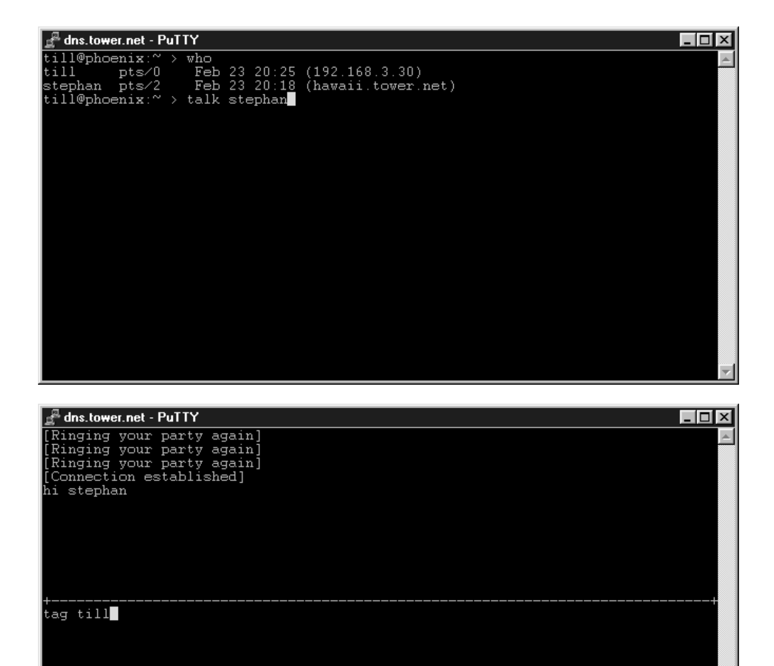

Abbildung 3.1: Der konventionelle Befehl »talk«

Neben diesen Chat-Einrichtungen gibt es weitere Chat-Programme und -Netzwerke, wie etwa ICQ von Mirabilis oder die verschiedenen Instant-Messaging-Systeme, die nicht immer Echtzeit-Dienste bereitstellen und in der Regel zusätzliche proprietäre Client-Software benötigen, um auf jedem teilnehmenden System installiert werden zu können.

Ein System hebt sich jedoch von der Menge ab. IRC (Internet Relay Chat) ist ein weit verbreitetes und seit langem verwendetes Chat-Protokoll, das von vielen Netzwerken benutzt wird, von denen einige Hunderttausende Benutzer gleichzeitig bedienen. Das IRC-Protokoll ist textbasiert, was von Nachteil ist, wenn Sie unter hoher Belastung arbeiten (denn lange Zeichenkettenbefehle erzeugen einen sehr viel höheren Datenverkehr als einzelne binäre Zeichen). Auf der anderen Seite vereinfacht es aber die Datenweiterleitung. Die meisten der heutigen IRC-Server unterstützen komprimierte Backbone-Verbindungen, die den Datenverkehr stark reduzieren.

IRC benötigt zwar für jedes teilnehmende System eine spezielle Client-Software, aber wir können diese Anforderungen zu unserem Vorteil nutzen. Warum sollten wir nicht die Client-Software auf der Serverseite bereitstellen

und sie abstrahieren, indem wir eine HTML-Schnittstelle benutzen und den Netzwerkzugriff für den Benutzer über einen HTML-Client ermöglichen? Dies gäbe uns die Kontrolle darüber, was der Benutzer tun kann (jeder Benutzer ist gezwungen, unseren HTML-Client zu verwenden) und was nicht. Zusätzlich stehen uns alle Vorteile eines bereits bestehenden Netzwerksystems zur Verfügung: zuverlässige Client-Software, ein bewährte Konzept, Hunderte von Werkzeugen usw. Wir könnten den Benutzern sogar erlauben, ihre eigene Client-Software zu benutzen. Diese Option werden wir in den meisten Fällen nicht nutzen, da wir ein »geschlossenes« Chat-Netzwerk erzeugen möchten. In einem geschlossenen Netzwerk kennen Sie alle Wege, über die ein Client auf Ihr Netzwerk zugreifen kann. Wenn Sie die Zugriffsmöglichkeiten auf spezielle Einstellungen beschränken, reduzieren Sie damit das Risiko, angegriffen zu werden.

Dies führt uns direkt zu der Frage, ob wir ein reales Protokoll wie IRC überhaupt brauchen. Oder würde es ausreichen, ein datenbankgesteuertes Protokoll mit einer Fernsynchronisierungsfunktion zu verwenden, um die erforderlichen Netzwerkfunktionen zu erhalten?

Fragen, wie diese, werden jedes Mal auftauchen, wenn Sie ein Programm planen, und sie tauchen häufig auf. Vergewissern Sie sich, dass Sie alle Aspekte bedacht haben und stellen Sie sicher, dass zu einem späteren Zeitpunkt der Entwicklung keine Fragen mehr offen sind. Hier und jetzt ist der Punkt, an dem Sie sich mit diesen Fragen befassen können. Später sind Sie vielleicht nicht mehr in der Lage, sie zu lösen (und müssen schließlich Ihr Projekt aufgeben. Ein gutes Projekt ist ein Projekt ohne Zweifel, ohne Unsicherheiten, ohne Inkonsistenzen und ohne unvorhergesehene Eventualitäten. Stellen Sie sicher, dass Sie nach der Planungsphase eine stabile und vollständig ausgewertete Situation gewährleisten können!

Lassen Sie uns damit zur Beantwortung der Frage zurückkehren: Brauchen wir ein offenes (und möglicherweise zu kompliziertes) Protokoll wie IRC oder sollten wir beim konventionellen Datenbankansatz bleiben? Die einfachste Methode, dies herauszufinden, ist gleichzeitig die logischste. Wägen Sie die Pro- und Contra-Argumente gegeneinander ab und wählen Sie die Option mit den besten Ergebnissen.

Wenn Sie IRC als Protokoll in Ihrem Chat-System einsetzen, müssen Sie aufgrund der Protokollverarbeitung mit erheblichen Komplikationen rechnen. Die Verarbeitung von Netzwerkprotokollen erfordert eine nicht-lineare Programmierung, was in PHP eigentlich nicht unterstützt wird. (Um auf Netzwerknachrichten reagieren zu können, brauchen wir ein ereignisbasiertes System.) Darüber hinaus benötigen wir ein Verfahren, um den Nachrichtenaustausch effizient zu bewerkstelligen, d.h. mit Nachrichten von einem Benutzer und für einen Benutzer umzugehen (was leider nicht immer auf dieselbe Art und Weise gehandhabt werden kann). Dieses Problem gibt es natür-

lich auch bei der datenbankgestützten Lösung. Diese hat allerdings nicht das Problem der Protokollhandhabung. Viele Datenbanken werden direkt von PHP unterstützt, und die meisten der anderen werden indirekt durch OBDC unterstützt. Um netzwerkfähige Chat-Boxen zu erhalten, brauchen wir lediglich ein Werkzeug, das Chat-Boxen synchronisieren kann (sofern Sie nicht einen zentralen Datenbank-Server einrichten möchten, auf den alle Boxen gleichzeitig zugreifen können.)

Für was würden Sie sich entscheiden?

Antwort: phpChat basiert auf IRC und zwar aus folgenden Gründen:

- Für eine Datenbank benötigten wir eine Art von »proprietärem« privatem Protokoll, das nicht mit anderen Standardsystemen kommunizieren könnte. In Zeiten plattformübergreifender Systeme wäre dies nicht von Vorteil.
- Eine IRC-Bibliothek, die gut funktioniert (nämlich phpIRC, siehe unter `www.phpwizard.net/phpIRC`), abstrahiert den Zugriff auf IRC-Netzwerke durch eine Reihe von leicht zu verwendenden API-Funktionen und setzt damit die IRC-Verwaltung in Bezug auf die Komplexität des Programms auf die gleiche Stufe wie die Datenbankverwaltung.
- Eine bereits vorhandene IRC-Serversoftware handhabt alle kleinen und großen Probleme der Benutzerverwaltung, des zuverlässigen Datenverkehrs, Routing usw. zwischen Netzwerken. Die Software gibt es bereits seit längerem und hat sich bewährt. Außerdem ist sie für alle Systemtypen verfügbar.
- IRC ist extrem skalierbar. Wenn Sie aufgrund von unvorhergesehenen Ereignissen auf dem Server A zu Spitzenzeiten eine zu hohe Belastung haben, schalten Sie einfach Server B hinzu und errichten dynamisch eine Serververbindung zu dem bestehenden Chat (IRC lässt dieses zu, und der Vorgang ist vollautomatisch). Damit haben Sie einen weiteren Server mit genügend freier Kapazität für zusätzliche Benutzer.

3.3 Grundlagen des IRC-Netzwerks

Nachdem wir einen Kommunikationsstandard für den Chat ausgewählt haben, möchten wir uns den Aufbau des IRC-Netzwerks genauer ansehen.

Idealerweise sollten Sie die IRC-Netzwerkgrundlagen, die in diesem Abschnitt besprochen werden, durchgearbeitet haben, bevor Sie sich für IRC entscheiden. Es ist immer schlecht, wenn Sie herausfinden, dass IRC die Struktur komplizierter macht, nachdem Sie sich dafür entschieden haben. Bisher sind wir jedoch von einem »allgemeinen Verständnis« über IRC-Netzwerke ausgegangen und haben die Programmplanung in den Vordergrund gestellt. Nach-

dem wir Ihnen die »richtige« Methode zur Verwendung des Programms (IRC) gezeigt haben, gehen wir jetzt auf die Einzelheiten ein, die Sie zur Ausführung des Plans benötigen.

IRC-Netzwerke unterscheiden zwischen Clients und Servern. Benutzer können am Netzwerk nur über eine spezielle Client-Software teilnehmen, die eine Verbindung vom Client zum Server herstellt. Alle Server sind im Netz über spezielle Serververbindungen miteinander verbunden. Die heutigen IRC-Server unterstützen nur hierarchische Strukturen. Dies bedeutet, dass es keine redundanten Verbindungen zu einem Server geben darf. Dadurch erhält das Netz eine baumähnliche Struktur, die einer Aufsplittung des Netzwerks gleichkommt, aber auch das Routing vereinfacht. Alle Server müssen lediglich alle ankommenden Daten an alle anderen Verbindungen schicken, ohne zu befürchten, dass sie einem Server redundante Informationen senden.

Jeder Server kann eine Reihe von Clients haben. Die maximale Anzahl hängt von der Anzahl der Verbindungen ab, die ein Server akzeptiert (natürlich wird sie auch durch die Netzwerkkapazität und die Serverbelastung begrenzt). Wie Abbildung 3.2 zeigt, kann jeder Server jeden anderen Server über mehr oder weniger Serversprünge erreichen, so dass jeder Server einfach alle ankommenden Daten an alle ausgehenden Verbindungen weiterleitet. Server C und Server F haben möglicherweise Clients, die am selben Kanal teilnehmen (Kanäle sind die Chat-Räume von IRC; Plätze, an denen sich Leute »treffen« und »unterhalten« können). In unserem Beispiel würde Server C Daten über die einzige ihm zur Verfügung stehende Verbindung schicken, nämlich zu Server B. Server B verteilt dann die Daten auf seine anderen Verbindungen, nämlich Server A und Server D. Server A hat keine weiteren Verbindungen, macht also nichts, aber Server D leitet die Daten an Server E weiter und der wiederum an Server F. Dies ist recht einfach zu realisieren, hat aber einen Nachteil: Wenn keiner der Clients, die mit Server A verbunden sind, an dem Datenverkehr über die Kanäle teilnimmt, über die Server C Daten schickt, werden alle Daten für Server A, die über diesen Kanal geschickt werden, einfach nur Bandbreite verschwenden.

RFC für IRC Ähnlich wie bei allen offenen Standards im Internet wurden die Grundlagen des IRC-Protokolls in einer RFC (Request For Comments) spezifiziert. Die RFC für IRC ist die RFC 1459, die Sie etwa unter `www.irchelp.org` herunterladen können. Diese Website enthält viele Informationen über IRC.

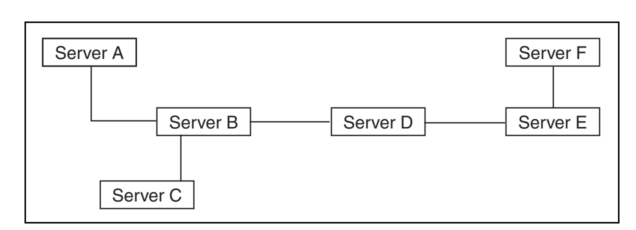

Abbildung 3.2: Eine IRC-Netzwerkstruktur

Dies ist eines der Hauptprobleme des Netzwerks, das »durch das Konzept begrenzt« ist. Der gesamte öffentliche Datenverkehr muss an alle Server gehen. Taucht dieses Problem jedoch tatsächlich unter den Bedingungen auf, unter denen wir unser IRC-Netzwerk installieren wollen? Sicher nicht, denn die Anzahl der Clients, mit der wir umgehen müssen, wird nie so groß sein, dass sie zu einem Problem wird – vorausgesetzt wir verwenden einen Standard-Host. In internen Netzwerken sollte dieses Problem überhaupt nicht auftreten.

Um die Gesamtanzahl der kritischen Verbindungen zu reduzieren, kann das Netzwerk so konzipiert werden, dass es seiner physikalischen Topologie folgt. Wenn ein Server mit einer höheren Kapazität als die anderen Server angeschlossen wird, kann er beispielsweise mehr Knoten bekommen als andere (viele Knoten an einen Server mit einem kleinen Backbone anzuhängen, macht nicht viel Sinn). Eine weitere Möglichkeit ist, das Routing an das Netzwerk anzupassen. Amerikanische Server haben beispielsweise ihren Sitz in den Vereinigten Staaten, deutsche Server in Deutschland usw. Frankfurt hat eine Überseeverbindung nach New York. Der IRC-Server in Frankfurt sollte daher mit dem New Yorker Server verbunden werden (entsprechend dem physikalischen Aufbau des Netzes). Man könnte es auch anders machen. Frankfurt könnte beispielsweise eine Verbindung nach Polen haben. Wenn Polen jedoch keine eigene Überseeverbindung hat, müsste der Datenverkehr, der von Frankfurt nach Polen weitergeleitet wird, einen anderen Weg finden, um den Ozean zu überqueren. Er würde über ein weiteres Land (oder zwei oder drei) geleitet werden, bis eine Überseeverbindung gefunden wird. Durch dieses zusätzliche Routing wird viel Bandbreite verschwendet. Daher wird stets versucht, die Struktur des IRC-Netzwerks so anzupassen, dass es bestmöglichst den zugrundeliegenden physikalischen Netzwerkaufbau unterstützt.

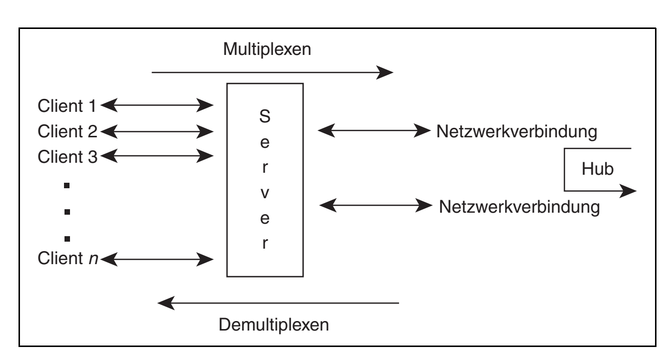

Abbildung 3.3: Netzwerkstruktur aus Serversicht

Diese konzeptuellen Probleme treten nur in den größten Netzwerken auf, die Zehntausende von Benutzern versorgen. Diese Netzwerke benötigen wirklich zuverlässige Verbindungen für ihre Backbones. Die typischen web-basierten

Chat-Räume oder Netzwerke haben meist nicht mehr als 1.000 Clients auf einmal, so dass nicht gleich zu Beginn ernsthafte Probleme zu erwarten sind. Um Komplikationen zu vermeiden, empfiehlt es sich trotzdem, ein Konzept zu entwickeln, um dieser Art von Problemen zu begegnen, die möglicherweise später auftreten. Aus Sicht des Servers stellt sich das Netzwerk wie in Abbildung 3.3 dar.

Die hier eingesetzte Struktur ähnelt einer Mischung aus Multiplexor, Demultiplexor und Hub. In der Richtung vom Client zum Netzwerk komprimiert der Server alle vom Client kommenden Daten und schickt diese zu den Netzwerkverbindungen. In der Gegenrichtung ermittelt er, welche vom Netzwerk kommende Informationen für welchen Client wichtig sind und schickt die Daten zu den entsprechenden Verbindungen. Alle vom Netzwerk ankommenden Daten, die zu den anderen Netzwerkverbindungen geschickt werden sollen, werden direkt weitergeleitet.

Prinzipiell ist dies die Einstellung, die wir für unser eigenes Chat-System brauchen. Lassen Sie uns nun überlegen, wie wir unser Ziel erreichen. Wir benötigen eine Server-Umgebung, die folgende Bedingungen erfüllt:

- Sie akzeptiert IRC-Netzwerkverbindungen.
- Sie akzeptiert IRC-Client-Verbindungen.
- Sie bietet eine web-basierte Benutzerschnittstelle.
- Sie ist möglichst einfach zu implementieren.

3.4 Einpassen der Anwendung in das Netzwerk

Wenn Sie bereits ein Konzept für die Entwicklung Ihres eigenen Servers in PHP hatten, überdenken Sie ihn noch einmal. Möglicherweise sind Sie fälschlicherweise davon ausgegangen, dass Sie zur Implementierung eines Chatservers auch einen Netzwerk-Server implementieren müssen. Dies ist zwar die Richtung, in die wir Sie tatsächlich führen wollten, aber Sie müssen ihn nicht selbst entwickeln. Es gibt bereits gut geschriebene Serversoftware für alle Systeme. Warum verwenden Sie also nicht einen der bestehenden Server und melden Ihren Server im Netzwerk als Client an? Alles was Sie tun müssen, ist, im Netzwerk eine weitere Abstraktionsebene hinzuzufügen, wie Abbildung 3.4 demonstriert.

Der PHP-Chatserver wird vom Webserver betrieben. Für jede Client-Verbindung, die er akzeptiert, erzeugt er eine Client-Verbindung zum IRC-Server. Auf diese Weise können wir sicherstellen, dass alle Daten, die für diesen Client ankommen, auch für diesen Client gedacht sind und für niemanden sonst.

Jeder Chat-Vorgang hat nur einen einzigen Benutzer und muss sich nicht um die anderen Benutzer kümmern. Die Benutzerkoordinierung, die Kontrolle des Datenverkehrs usw. wird vom IRC-Server übernommen, für den wir einfach einen der frei verfügbaren Server verwenden.

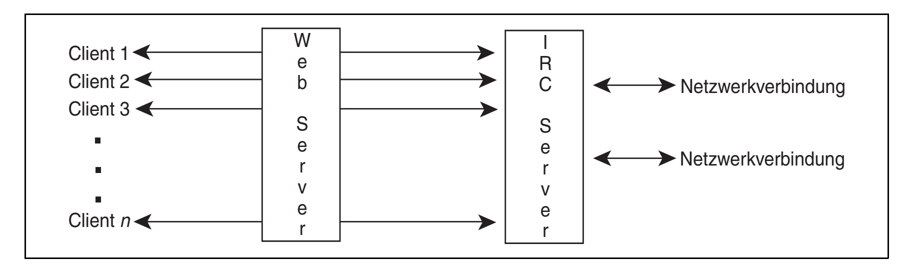

Abbildung 3.4: phpChat als Abstraktionsebene für den Server

Diese Vorgehensweise hat den zusätzlichen Vorteil, dass dieses Chatserver-Programm auch als sicherer Gateway für das IRC-Netzwerk verwendet werden kann (siehe auch Abbildung 3.5). Viele private und Firmennetzwerke sind hinter Firewalls geschaltet, welche die IRC-Ports filtern. Da dieser Chat nur per HTTP mit den Clients (die nicht gefiltert werden) kommuniziert, benötigt nur der Chatserver eine offene Verbindung zum IRC-Server.

Daher beschränken wir uns an dieser Stelle auf die Installation der Clientsoftware, die ansonsten auf der Benutzerseite unseres Webservers erforderlich wäre. IRC kennt alle Befehle, die zum Einrichten eines leistungsfähigen Chats notwendig sind, und alles was das Netzwerk betrifft, kann über ein gängiges Standardserver-Programm gelöst werden. Wenn unsere Schnittstelle alle IRC-Funktionen auf komfortable Weise unterstützt, haben wir unsere Arbeit erledigt.

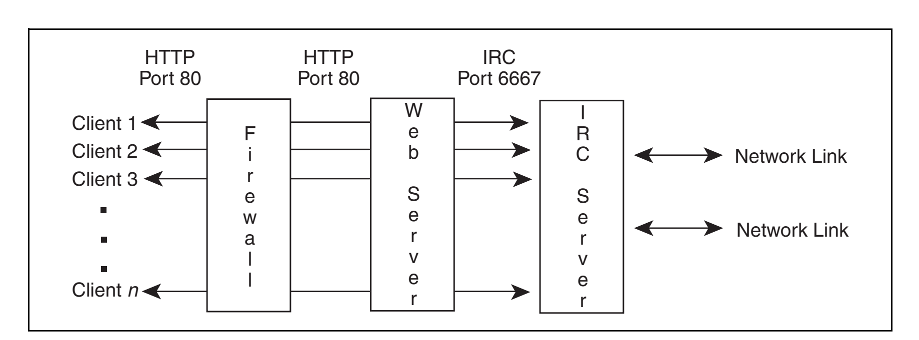

Abbildung 3.5: phpChat als sicherer IRC-Gateway

3.5 Schnittstelle zum Netzwerk

Wie wir bereits erwähnt haben, bedeutet der Einsatz vom IRC einen Mehraufwand für die Verarbeitung. Eine komplette Protokollroutine einzuhacken, die eine Schnittstelle zum IRC bereitstellt, wäre ein wenig zu kompliziert. Wir haben IRC aber auch deshalb einer datenbankbasierten Lösung vorgezogen, weil es bereits eine API gibt, die diese Aufgabe für uns übernimmt.

Informieren Sie sich über den Markt! Für jedes Programmierprojekt ist es immens wichtig, welche Teile bereits von anderen entwickelt wurden und welche es noch nicht gibt. Erfinden Sie das Rad nicht neu! Besonders bei kommerziellen Projekten kann es sich durchaus auszahlen, fremde absturzsichere Lösungen für bestimmte Aufgaben zu kaufen und sie nicht selbst zu konzipieren und zu entwickeln. Letzteres ist manchmal teurer und zeitaufwendiger. Außerdem werden externe Lösungen in der Regel ständig verbessert. Dieser Vorgang ist absolut unabhängig von der Entwicklung Ihres eigenen Projekts. Wenn Sie ein Upgrade von einer externe Firma erhalten, müssen Sie nur noch den entsprechenden Teil Ihrer Anwendung ersetzen. Auf diese Weise können Sie einige Teile Ihres Programms aktualisieren, ohne in die Änderungen selbst Arbeit zu stecken. Wenn Sie zudem bereits bestehende Bibliotheken verwenden, stützen Sie Ihr Projekt automatisch auf allgemein üblichen, standardisierten APIs; dies ist immer von großem Vorteil ist.

Auf der anderen Seite kann sich die Bindung an fremde Produkte natürlich auch als Nachteil erweisen, wenn der Hersteller sein Produkt nicht auf dem neuesten Stand hält oder wenn es Fehler enthält, die nicht korrigiert werden.

Nach unserer Erfahrung haben sich Open-Source-Produkte als erfolgreichste externe Teile erwiesen. Sie werden in der Regel recht schnell verbessert und erweitert und stützen sich auf allgemeingültige und offene hochleistungsfähige Standards.

Übung: Suchen Sie nach Anwendungen / Bibliotheken, die in PHP geschrieben sind und IRC verwenden. Vergleichen Sie diese in Bezug auf Design, Flexibilität und Gebrauchsfreundlichkeit. Die Implementierung ist zwar auch interessant, sollte aber nicht Ihr Hauptinteresse sein. Der wichtigste Teil der Entwicklung ist das Konzept. Wenn dieses steht, ist die tatsächliche Realisierung recht einfach (auch wenn viele Programmierer dies anders sehen).

Wir haben für dieses Projekt die Bibliothek phpIRC ausgewählt (`www.phpwizard.net/phpIRC`). Sie bietet folgende Vorteile:

- Sie ist leicht zu verwenden.
- Sie stellt eine vollständige und leistungsfähige API dar.
- Sie verwendet eine ereignisbasierte Verarbeitung.

Die Verwendung der ereignisbasierten Verarbeitung ist hier von besonderem Interesse. Dieses Verfahren wird üblicherweise in den konventionellen Anwendungen implementiert. So sind z.B. alle Windows-Programme ereignisbasiert. Ereignisbasierte Programme laufen in einer Endlosschleife, wobei sie darauf warten, dass etwas (ein Ereignis) passiert. Ein Ereignis kann u.a. eine Benutzereingabe, eine Bewegung mit der Maus, ein Netzwerkereignis (ankommende Pakete) usw. sein. Sobald ein Ereignis angekündigt wird, verlässt das Programm seine Hauptschleife und sucht nach einer Prozedur, die dieses Ereignis bearbeitet. Alle Prozeduren, die das gerade aufgetretene Ereignis abarbeiten möchten, werden mit den angegebenen Parametern des Ereignisses aufgerufen (z.B. Paketdaten des ankommenden Netzwerkverkehrs).

Bei der »konventionellen« Programmierung würde ein ankommendes Ping beispielsweise wie in Listing 3.1 verarbeitet:

```
again:

wait_for_network_data();

if(incoming_data == ping)
{
    send_pong();
    update_traffic_counter();
}

goto again;
```

Listing 3.1: Pseudocode für die Handhabung eines Pings

Dieser Programmcode wartet, bis er Daten vom Netzwerk erhält. Danach versucht er herauszufinden, ob die Daten ein Ping waren. Wenn ja, schickt der Code ein Pong zurück und aktualisiert aus statistischen Gründen einen Datenverkehrszähler. Danach springt er an die Stelle zurück, an der es begann. Stellen Sie sich nun einen Code mit Hunderten von Ereignissen vor, von denen einige möglicherweise von anderen abhängen, einige nicht, andere nur unter bestimmten Umständen – eine Horrorvorstellung!

Ereignisbasierte Programmierung macht es deutlich einfacher, wie Listing 3.2 zeigt:

```
event_handler ping()
{

    send_pong();

}
```

```
event_handler incoming_data()
{

    update_traffic_counter();

    case of ping: handle_event(ping);
}

while(not_done())
{

    wait_for_event();

    case of network_data: handle_event(incoming_data);

}
```

Listing 3.2: Ereignisbasierter Pseudocode für die Handhabung eines Pings

Dieser Code ist länger, aber auch klarer. Die Hauptschleife wartet, dass ein Ereignis passiert. Wenn sie herausfindet, dass ein Ereignis stattgefunden hat und es durch ankommende Netzwerkdaten ausgelöst wurde, fertig sie dieses Ereignis mithilfe der zentralen Prozedur `handle_event()` ab. Diese Funktion bestimmt dann eine Routine für das Ereignis und ruft sie auf. Die Routine aktualisiert ihrerseits den Datenverkehrszähler und startet ein weiteres Ereignis, wenn das erste Ereignis ein Ping war. Nachdem dieses Ereignis wiederum mit `handle_event()` abgefertigt worden ist, wird ein Pong gesendet.

Alternativ könnten sich sowohl `ping()` als auch `incoming_data()` sich bei dem Ereignis `"incoming_data"` anmelden. Die Erstellung zweier verschiedener Ereignisse ermöglicht jedoch eine größere Vielfalt an Ereignissen und erlaubt somit eine detailliertere, zielgerichtete Verarbeitung.

Es ist zunächst seltsam, sich mit der ereignisbasierten Verarbeitung von Informationen vertraut zu machen (sie funktioniert ähnlich wie eine Finite-State-Maschine), aber sie hat viele Vorteile:

- Der Anwendung wird eine modulare Struktur aufgezwungen. Jedes Modul arbeitet unabhängig von den anderen Modulen und kann leicht geändert, ausgetauscht oder erweitert werden.
- Jeder Teil des Programms kann ein beliebiges Ereignis auslösen und daher jede Art von Reaktion in der Anwendung erzwingen (mit anderen Worten: Sie können jeden Teil Ihres Programmcodes von jedem anderen Teil Ihres Codes aus kontrollieren).

- Alle Daten können von einem zentralen Punkt des Programms transparent an alle Empfänger abgesetzt werden. Sie müssen sich nicht darum kümmern, Strukturen manuell zu kopieren und umzuwandeln. Die einzelnen Ereignisroutinen stellen selbst sicher, dass sie ihre Daten erhalten.
- Neue Programmcode lässt sich extrem leicht in das Programm einfügen, indem Sie einfach eine Prozedur schreiben, die sich selbst bei dem entsprechenden Ereignis registriert.

Sobald der große Rahmen für die Ereignisabwicklung erstellt wurde, kann die gesamte Anwendung generiert werden, indem Sie Routinen, Routinen und noch einmal Routinen schreiben.

Machen Sie sich mit den Verfahren vertraut, die zur Implementierung von Finite-State-Maschinen erforderlich sind. Diese sind für die Programmierung und allgemeine Informationsverarbeitung elementar.

Glücklicherweise enthält phpIRC bereits einen Rahmen für die Ereignisabwicklung, so dass uns diese Programmierung für unser Projekt erspart bleibt.

3.5.1 Struktur der Schnittstelle

phpIRC bildet den IRC-Client-Teil des Programms. Es ist für die gesamte Netzwerkkommunikation verantwortlich. Dies bedeutet auch, dass es stets die Kontrolle behalten muss, um rechtzeitig auf Netzwerknachrichten reagieren zu können. Wenn die nachrichtenverarbeitenden Funktionen von phpIRC nur gelegentlich aktiviert würden, könnte ein sicherer, zuverlässiger und schneller Datenaustausch nicht mehr gewährleistet werden. Aus diesem Grund erzwingt phpIRC eine bestimmte Programmstruktur, wie Abbildung 3.6 zeigt.

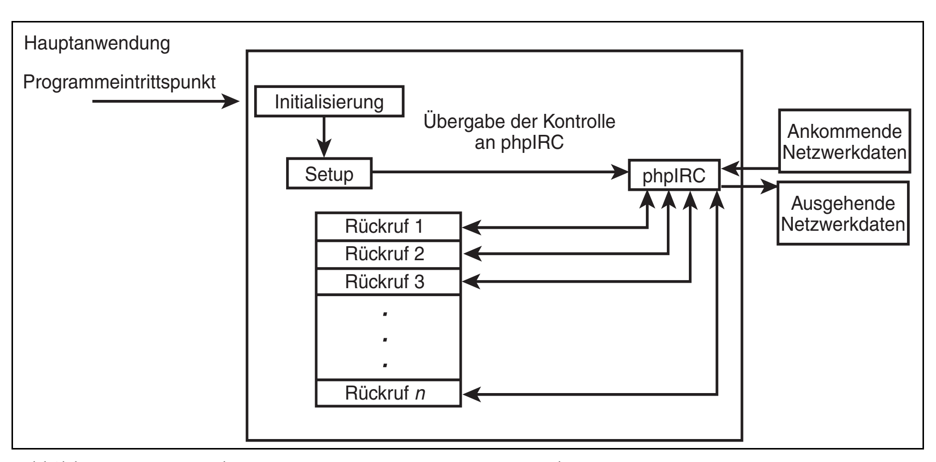

Abbildung 3.6: Von phpIRC erzwungene Programmstruktur

Nach der Initialisierung und dem Setup muss das Programm die Kontrolle an phpIRC übergeben. phpIRC springt dann an den Start seiner Hauptereignisschleife und wartet auf ein Ereignis. Während des Setups muss das Programm für jedes Ereignis, das es verarbeiten möchte (z.B. ankommende private Nachrichten, ankommende Server-Nachrichten usw.), Rückrufe registrieren. Diese Nachrichten sind für das Programm die einzige Möglichkeit, die Kontrolle zurückzubekommen. phpIRC übergibt dann alle Ereignisse an alle Funktionen, die sich in der Bibliothek registriert haben. Diese Funktionen können wiederum in eine weitere freie Schleife in phpIRC eintreten, die auf ein weiteres Ereignis wartet, oder sie können die API von phpIRC nutzen, um bestimmte Aktionen im Netz durchzuführen (private Nachrichten senden, Kanäle verbinden / verlassen usw.).

Dieses grundlegende Basislayout ermöglicht bereits eine Downstream-Kommunikation, so dass phpIRC in der Lage ist, Nachrichten von anderen Benutzern zu empfangen. Es können tatsächlich andere Personen mit Ihrem Skript »reden«.

Hinweis: *Downstream* bedeutet, vom Netzwerk zum Benutzer. Upstream ist das Gegenteil, d.h. vom Benutzer zum Netzwerk.

Übung: Strukturieren Sie eine Downstream-Schnittstelle, die phpIRC-Funktionen verwendet. Realisieren Sie diese auf dem Papier und machen Sie sich mit der API von phpIRC vertraut. Generieren Sie dann eine einfache Downstream-Schnittstelle, die über IRC Protokoll führt und alle Nachrichten von einem angegebenen Kanal anzeigt.

3.5.2 Downstream-Kommunikation

Da Chatten eine Echtzeit-Angelegenheit ist, was bedeutet, dass die Prozesse ablaufen, während Sie Gespräche führen und Antworten sofort bereitstellt, möchten wir die Schnittstelle nicht mit einer Latenzzeit versehen. Latenzzeit beschreibt die Reaktionszeit der Schnittstelle, z.B. die Zeit zwischen dem Moment, da der Benutzer die Enter-Taste drückt um eine Nachricht abzuschicken, bis zu dem Augenblick, an dem diese im Chat-Fenster erscheint. Eine Latenzzeit von weniger als einer Sekunde mag zwar objektiv gesehen eine sehr kurze Wartezeit sein, für den Benutzer erscheint sie jedoch extrem lang und ärgerlich. Ankommende Nachrichten sollten daher sofort angezeigt werden (oder zumindest so bald wie möglich). HTTP ist jedoch ein zustandsloses Protokoll, das sofortige Aktualisierung von Seiten nur ermöglicht, wenn ein vollständiges Dokument neu geladen wird. Natürlich gibt es mehrteilige Dokumente und eine automatische Auffrischung, aber bei jeder dieser Aktionen entsteht ein lästiges Flimmern wenn eine Seite neu geladen wird, die

Datenbank muss für die Ausgabe zwischengespeichert werden, und durch die ständigen Neuverbindungen und Datenübertragungen vom Webserver entstehen Verzögerungen.

Eine Lösung hierfür ist »Streaming HTML«. Diese Option wird zwar nirgendwo offiziell unterstützt, funktioniert aber trotzdem. Das Skript, das für die Schnittstellenausgabe verantwortlich ist, läuft einfach in einer Endlosschleife leer und beendet die HTML-Seite, die der Browser erhält, nicht. Wenn etwas an den Benutzer geschickt werden muss, wird es ausgegeben und sofort aus dem Zwischenspeicher des Servers gelöscht. Auf diese Weise gibt der Browser ständig Informationen aus und zeigt stets die aktuellsten Daten. Ein Problem birgt dieser Ansatz jedoch: HTML-Entitäten lassen sich nicht einfach so darstellen. Sie können beispielsweise die Zeilen in einer Tabelle nicht nacheinander ausgeben, da der Browser alle Zeilen und Spalten einer Tabelle kennen muss, um die endgültige Größe der Tabelle zu bestimmen. Solange Sie sich darauf beschränken, Textzeilen nacheinander auszugeben, und Tabellen nur verwenden, wenn Sie diese als Ganzes darstellen können, funktioniert alles recht gut.

Kleine Umwege, wie Streaming-HTML, sind übliche Tricks, die Sie kennen sollten. Halten Sie sich über solche Sachen stets auf dem Laufenden.

Streaming-HTML hat eine weitere Konsequenz, die einige als Nachteil, andere als Vorteil betrachten: Da die Client-Verbindung offen bleibt, muss es immer einen Serverprozess geben, der diese Verbindung verwaltet. Dies bedeutet, dass für jeden Client mindestens ein Webserver-Prozess vorhanden sein muss, der nur für diesen Client ausgeführt wird. Der Vorteil dieses Verfahrens besteht darin, dass es keine Mehrbelastung »pro Treffer« gibt. Wenn der Client ein Dokument anfordert, muss für gewöhnlich ein neuer Prozess erzeugt werden. Das Skript, das dieses Dokument generiert, muss geladen, analysiert und ausgeführt werden. Abschließend werden die Daten gesendet. Da der Serverprozess nun im Speicher bleibt, muss das Öffnen eines neuen Prozesses und Laden eines Skriptes sowie seine Interpretation für jeden Client nur einmal durchgeführt werden. Auf Websites, die ansonsten Hunderte von Treffern pro Sekunde hätten, ist dies ein klarer Vorteil. Jeder Prozess bleibt jedoch im Speicher resident und benötigt eigenen RAM. Auf Intel-x86-Systemem, die mit Linux, Apache und PHP 4.0 ausgestattet sind, können solche Prozesse jeweils bis zu 2 MB groß sein. Ein kleiner Server mit minimalem RAM würde bald vom Swap-Speicher ausgeführt – was seinen sicheren Tod bedeutet.

Haben Sie schon über die Folgen von residenten Prozessen nachgedacht? Wenn nicht, denken Sie daran, es das nächste Mal zu tun! Werten Sie jede Situation komplett aus.

Hinweis: Der Swap-Speicher ist ein virtueller Speicher, der den RAM erweitern soll, den physikalischen Speicher auf einem Rechner. Der Swap-Speicher wird auf einer Festplatte abgelegt, was extrem langsam ist. Wenn der physikalische Speicher vollständig belegt ist, beginnen moderne Betriebssysteme damit, neuen Speicher in dem langsamen Swap-Speicher zu belegen. Sobald viele Clients gleichzeitig auf denselben Chatserver zugreifen und dabei den gesamten physikalischen Speicher verbrauchen, werden Daten in den Swap-Speicher geschrieben. Nun muss das Betriebssystem ständig Teile des RAM gegen Teile aus dem Swap-Speicher austauschen (da Programme nicht vom Swap-Speicher aus gestartet werden können). Dies setzt einen »Teufelskreis« in Gang: Das Betriebssystem bemerkt, dass ein Prozess im Swap-Speicher ausgeführt werden muss, und lädt es in den RAM, muss dafür aber einen anderen Prozess aus dem RAM in den Swap-Speicher auslagern. Es führt den Prozess im RAM aus, bevor es feststellt, dass der alte Prozess (der sich nun im Swap-Speicher befindet) ausgeführt werden muss. Also transferiert es die Daten wieder in den RAM usw. Auf diese Weise können Sie einen Server sehr leicht zu Absturz bringen, so dass er neu gestartet oder vom Netz genommen werden muss. Dies ist übrigens eine übliche »Dienstverweigerungs-Attacke« ähnlich jener, die Yahoo! und andere dieses Jahr bereits über sich ergehen lassen mussten.

3.5.3 Upstream-Kommunikation

Als Nächstes beschäftigen wir uns mit der Upstream-Kommunikation, d.h. der Weiterleitung von Benutzereingaben zum Netzwerk. Jetzt kommt der harte Teil. Wir können Daten zum IRC-Netzwerk nicht von jedem Prozess aus senden. Warum nicht? Weil IRC ein zustandsensitives Protokoll ist, ist die Kommunikation an eine bestimmte Client-Verbindung gebunden. PHP lässt keine Übernahme fremder Sockel von anderen Prozessen zu. Der Hauptprozess, der auch die Downstream-Kommunikation übernimmt (der Prozess, der als IRC-Client agiert) wird isoliert von allen anderen Prozessen ausgeführt. Nun ist die Frage, wie wir eine Tür öffnen können, um Daten an den Haupt-Client weiterzuleiten?

Wie würden Sie eine Upstream-Kommunikation realisieren? Versuchen Sie zumindest einen theoretischen Ansatz zu entwickeln. Zeichnen Sie ein Datenflussdiagramm auf Papier. Wenn Sie es noch nicht getan haben, schreiben Sie mindesten drei Möglichkeiten für den Laufzeit-Datenaustausch auf.

Der Downstream-Prozess muss permanent laufen und darf nicht beendet werden. Wir können ihn nicht einfach über ein POST oder ein GET für die Datenweitergabe neu aufrufen, denn dies würde bedeuten, einen weiteren Prozess zu starten, der wiederum neu angemeldet, neu eingerichtet usw. werden müsste. Ein solcher Ansatz würde zu ständigen Anmelde- und Abmelde-

vorgängen führen, die in einem Chat extrem störend wären. Außerdem würde es zu Datenverlusten führen, da in der Zeit zwischen einem Logout und einem Login viele Nachrichten übertragen werden könnten (die für einen neu angemeldeten Client nicht sichtbar wären).

Der Chat könnte auf einen einzelnen Bot basieren, der die ganze Zeit online bleibt und alle Nachrichten für alle Benutzer in einer Datenbank speichert. Die Benutzerschnittstelle müsste dann nur alle relevanten Daten aus der Datenbank auslesen. Zwei Probleme sprechen jedoch gegen diese Möglichkeit: (a) Der Chat wäre in erster Linie datenbankgestützt (was wir vermeiden wollten) und (b) die Clients wären für die anderen IRC-Clients sichtbar, da der Bot der einzige »echte« Client im Netz wäre. Damit wäre die Benutzung eines IRC-Netzwerks jedoch überflüssig.

Wir brauchen also mindestens zwei unabhängige Prozesse: einen, der die IRC-Kommunikation übernimmt und nicht unterbrochen werden kann, und einen weiteren, der alle ankommenden Nachrichten vom Benutzer annimmt. Außerdem brauchen wir eine Art von »Behälter« als Schnittstelle zwischen diesen beiden Prozessen. Abbildung 3.7 demonstriert dieses Problem.

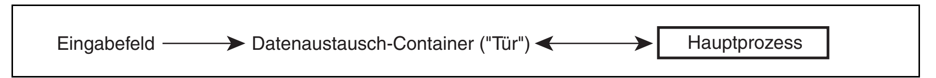

Abbildung 3.7: Upstream-Kommunikation

Die Situation lässt sich mit einem Autorennen vergleichen. Der Fahrer auf der Piste ist der »Haupt-Client« und das Rennteam in der Box entspricht dem Benutzereingabefeld. Der Fahrer ist an das Rennen gebunden, in dem er sich befindet. Er kann nicht einfach die Bahn verlassen und anhalten, um nachzusehen, was los ist. Wenn ihn das Rennteam per Flagge zum Halt in der Box auffordert, kommunizieren sie mit ihm, indem sie ihm ein Signal geben, nach der nächsten Runde die Fahrt zu unterbrechen.

Die Kommunikation besteht hier (wenn man den Funk einmal beiseite lässt) aus dem Signal, das gegeben wird, sobald ein Fahrer die Ziellinie überquert. Dieses Signal funktioniert als »Schnittstelle« zum Fahrer. Im Prinzip ist dies genau das, was wir auch tun müssen, um unserem Hauptprozess ein Signal geben. Da der Hauptprozess ereignisbasiert ist, haben wir häufig die Möglichkeit, die Kontrolle über das Programm zu übernehmen und die gewünschte Aktion durchzuführen. Wir können somit eine Verarbeitungsroutine installieren, die häufig nach einem Signal von außen Ausschau hält. Das Verfahren, das daraus besteht, periodisch anzuhalten und zu prüfen, ob ankommende Daten vorhanden sind, nennt sich Polling. Es ist die bevorzugte Methode in phpChat. phpIRC unterstützt freie Rückrufe. Diese werden immer dann aufgerufen, wenn phpIRC nichts zu tun hat und einfach darauf wartet, dass etwas auf dem Netzwerk passiert. Wir können nach einem Signal

suchen, indem wir diesem Ereignis eine Verarbeitungsroutine zuteilen. Wie sollen wir jedoch etwas signalisieren? Im Prinzip ist das sehr einfach, wenn wir eine der folgenden Methoden verwenden:

- Einen Flag in einer Datenbank setzen
- Eine Sperrdatei im Dateisystem erzeugen
- Synchronisierungsbefehle verwenden
- Einen Flag im Shared Memory setzen

Im Prinzip sind dies die Methoden, die wir in PHP einsetzen können um eine »Nachricht zu hinterlassen«. In den folgenden Abschnitten gehen wir auf die einzelnen Verfahren näher ein.

Pipes können für die prozessübergreifende Kommunikation nicht verwendet werden, da für eine Pipe zwei Prozesse gleichzeitig ausgeführt werden müssen. Unsere Situation erfordert, dass ein ständig laufender Prozess von anderen kurzfristigen Prozessen angesprochen wird.

Hinweis: Natürlich gibt es auch exotischere Verfahren, wie etwa das Verschicken von Emails zwischen Prozessen. Wir haben dies zwar schon gesehen, möchten aber hier nicht auf diese Option eingehen, da die Nachteile davon klar auf der Hand liegen sollten.

Setzen eines Flags in einer Datenbank

Das Setzen eines Flags in einer Datenbank ist wahrscheinlich die de-facto-Standardmethode in PHP: Verbindungen mit einer Datenbank herstellen, einige Daten darin hinterlassen und sie von anderen Prozessen weiterverarbeiten lassen. Diese Methode ist sehr einfach zu installieren und auf allen Systemen verfügbar, hat jedoch einen Nachteil. Wissen Sie von welchem Nachteil wir sprechen?

Der Nachteil entsteht nicht durch die schreibende Anwendung der Datenbank (dem Prozess der Benutzern Nachrichten einfügt), sondern kommt von dem lesenden Teil der Datenbank (dem Hauptprozess, der alle Benutzernachrichten von der Datenbank einliest). Um ein gutes »Chat-Gefühl« zu erhalten, brauchen wir so geringe Latenzzeiten wie möglich und damit sehr gute Antwortzeiten. Die Antwortzeit ist bei web-basierten Chats von großer Bedeutung, denn sie vermittelt dem Benutzer das Gefühl mitten drin zu sein. Wenn die Nachrichtenübertragung langsamer und langsamer wird, sind Benutzer schnell frustriert und geben auf. Unser Test ergab, dass eine Latenzzeit von mehr als einer Sekunde zu lang ist. Um unter diesem Wert zu bleiben, sollte die Abruffrequenz, mit welcher der Hauptprozess Nachrichten von der Datenbank einliest, sehr hoch sein. Bei phpChat liegt der Standardwert bei 0,5 Sekunden (zwei Überprüfungen pro Sekunde). Sobald ein Chatsystem viele

Clients verwalten muss, steigt die Belastung der Datenbank, die mehr und mehr Ressourcen belegt. Bei ungefähr 40 bis 50 Anfragen pro Sekunde brauchte unser Testserver etwa ein Drittel seiner Verarbeitungszeit nur für die Ausführung der Datenbankanfragen. Auch wenn dies ein Ausschlusskriterium für das Datenbanksystem darstellt (dieses sollte in der Lage sein, weitaus mehr Anfragen zu verarbeiten), gibt es hier offensichtlich einen Optimierungsbedarf, und dies ist nicht die ideale Einstellung.

Erstellen von Sperrdateien

Unser zweiter Gedanke war, dass ein Dateisystem effizienten sein könnte, wenn die Datenbank für die prozessübergreifende Kommunikation zu viele Ressourcen belegt.

Das Dateisystem hat jedoch das Rennen klar verloren. Auch hier war die schreibende Anwendung nicht das Problem. Die Sperrdateien ließen sich problemlos erzeugen. Um zu ermitteln, ob eine Sperre gesetzt war, musste jedoch `clearstatcache()` sehr häufig aufgerufen werden, um zuverlässig herauszufinden, ob eine Sperrdatei inzwischen gelöscht wurde oder noch vorhanden war. `clearstatcache()` hat die Systemleistung sehr stark beeinträchtigt, so dass wir diese Option nicht näher in Betracht zogen. Der Chat funktioniert nur mit einem Viertel der Leistung, die er mit dem datenbankgestützten Ansatz erbrachte.

Schaffen Sie sich eigene Bezugspunkte. Schreiben Sie einige Testskripte, mit denen Sie auf die Datenbanken und das Dateisystem mit einer hohen Frequenz zugreifen. Schreiben Sie Ihre Resultate nieder und vergleichen Sie diese. Dies ist immer eine gute Methode, um Datenaustauschverfahren zu testen. Vertrauen Sie niemals auf die theoretischen Beschreibungen dessen, zu was das System fähig sein soll. In der Praxis funktionieren viele Dinge ganz anders.

Verwendung von Semaphoren

Natürlich haben Sie den Grund für die schlechte Leistung in den beiden vorangegangenen Ansätzen schnell erkannt.

Was sind die Gründe? Finden Sie es heraus und schreiben Sie es nieder. Suchen Sie nach den kritischen Punkten. Dies ist wichtig, wenn Sie das Verfahren später optimieren wollen. Eine Kette ist nur so stark wie ihr schwächstes Glied. Und ein Programm ist nur so schnell wie seine langsamste innere Schleife. Die Methode, um diese Engpässe zu ermitteln, heißt *Profiling*; sie ist extrem wichtig.

Bei der Verwendung einer Datenbank ist der Engpass die Datenbank. Die Zeit, die Datenbank aufzurufen, eine (relativ kleine) Anfrage auszuführen, das Ergebnis einzulesen und zu ermitteln, was als Nächstes zu tun ist (der

sogenannte Overhead), ist im Vergleich zu dem Ergebnis, das wir erhalten, sehr lang. Mit anderen Worten: Wir verwenden ein riesiges Softwaresystem, das für komplexe Datenspeicherung ausgelegt ist, um einfache boolesche Werte auszutauschen. Wenn eine Datenbank für irgendetwas nicht konzipiert wurde, dann dafür. Kein Wunder also, dass sie keine optimale Leistung erbrachte. Der Engpass ist der Overhead, d.h. die Zeit, die für den ständigen Verbindungsauf- und -abbau benötigt wird.

Das Dateisystem erbrachte eine schlechte Leistung, weil es nicht für diese Verwendung konzipiert wurde. Darüber hinaus gab es weitere Begrenzungen: PHP hat keine optimalen Methoden für den Dateisystemzugriff. Um die Existenz einer Datei zu überprüfen muss, ständig der Cache-Speicher gelesen und gelöscht werden – und wieder entsteht ein großer Overhead für eine einfache Aufgabe.

Warum verwenden wir also nicht etwas vollkommen anderes? Wir sind sicher nicht die Ersten, die eine prozessübergreifende Kommunikation verwalten müssen. Hierfür müssen andere schon gute Lösungen entwickelt haben. Damit kommen wir zur nächsten Möglichkeit: *Semaphore*.

Semaphore machen genau das, was wir wollen: Sie funktionieren als Signale. Es handelt sich hierbei um Zähler, die im gemeinsam genutzten Speicher, dem sog. Shared Memory abgelegt werden. Sie können ein Semaphor »erfassen« und damit seinen Zähler hochsetzen oder ihn »freigeben«, womit sein Zähler herabgesetzt wird. Außerdem können Sie darauf warten, dass ein Semaphor frei wird, d.h. dass sein Zähler zurück auf Null fällt. Dieses Verfahren hat jedoch einen Nachteil: Semaphore dienten ursprünglich dazu, Ressourcen zu sperren, um eine Art von Zeitplanung zu erhalten. Prozesse konnten so darauf warten, bis auf einem Gerät Zeit verfügbar war oder Ähnliches. In der Zeit, in der Sie darauf warten, dass ein Semaphor frei wird, wird der wartende Prozess schlafen gelegt und kann keine anderen Aufgaben übernehmen. Wenn nun der Hauptprozess darauf wartet, dass das Benutzereingabefeld eine neue Nachricht ankündigt, kann er den ankommenden Netzwerkdatenstrom nicht verarbeiten, weil er gerade »schläft«.

Trotzdem ist das noch kein Grund jetzt aufzugeben. Es gibt noch andere Lösungen.

Setzen von Flags im Shared Memory

Das Shared Memory ähnelt den Semaphoren, ist aber vielseitiger. Hierbei handelt es sich um einen Speicher, der für jeden Prozess in einem System verfügbar ist. Multitasking-Systeme sind in der Regel so konzipiert, dass jeder Prozess aus Sicherheitsgründen vollständig unabhängig von den anderen Prozessen ausgeführt wird. Mehrere Prozesse können dieselben Daten benutzen, indem sie spezielle Speicherblöcke, sogenannte Shared Memory-Blöcke

einrichten und eine Verbindung zu diesen herstellen. Diese Blöcke können Variablen enthalten (oder jede andere Art von Daten, wobei PHP nur die Speicherung von konkreten Variablen unterstützt).

Genau dies wollen wir: die Möglichkeit einen booleschen Wert an einer Stelle im Speicher abzulegen, an der jeder Prozess nachsehen kann. Da das Shared Memory nur im RAM agiert, ist er extrem schnell und benötigt fast keine zusätzlichen Kapazitäten. Bei dieser Option sucht jeder Chat-Prozess seine eigene Variable im Shared Memory und gibt nur dann eine Anfrage an die Datenbank aus, wenn er die Variable findet, die vom Benutzereingabefeld gesetzt wurde.

Warum basiert der Datenaustausch letztendlich doch auf einer Datenbank? Finden Sie einige Antworten.

Die Datenbank wird hauptsächlich aus einem Grund verwendet. Das Shared Memory wird standardmäßig nicht von PHP unterstützt. Sie müssten in PHP dafür extra eine Unterstützung programmieren. Viele Leute, die auf einen PHP-fähigen Server zugreifen, haben jedoch nicht die Möglichkeit, PHP neu zu programmieren, da sie nur Speicher auf dem Server gemietet und so nicht die entsprechenden Rechte haben. Möglicherweise benötigen aber auch andere genau diese vorhandene PHP-Einstellung. Wenn wir die Datenbank als endgültigen Datenaustauschpfad beibehalten, können wir das Shared Memory als optionale Optimierung verwenden. Diejenigen, die ihn nicht nutzen können, deaktivieren ihn einfach und haben trotzdem noch eine funktionierende Version des Chatservers, auch wenn sie sich mit einer sub-optimalen Lösung zufrieden geben müssen.

Wenn Sie ein Programm erstellen, das für eine weite Verbreitung gedacht ist, bedenken Sie, dass nicht jeder dieselbe Einstellung hat wie Sie und möglicherweise nicht in der Lage ist, Ihre ganz spezielle Einstellung wieder herzustellen. PHP arbeitet zwar zu 99% systemunabhängig, aber einige Dinge hängen eben doch vom System ab. Wägen Sie jedes Mal sorgfältig ab, ob die Festlegung des Systems auf bestimmte Gegebenheiten einen möglicherweise riesigen Verlust von Kunden wert ist.

3.5.4 Schnittstelle zum Benutzer

Nachdem wir alle für den Datenaustausch problematischen Aspekte beseitigt haben, ist die eigentliche HTML-Schnittstelle zum Benutzer trivial. Wir wissen, wie wir mit Eingaben des Benutzers umgehen müssen und wie die Netzwerkkommunikation zu verwalten ist. Das letzte »Problem« besteht darin, die erzeugte Ausgabe für den Benutzer auf eine bequeme Art und Weise zu verpacken. HTML bietet nur eine Möglichkeit, um mehrere Fenster im Browser unabhängig voneinander anzuzeigen: Frames. Die Schnittstelle besteht in der Regel aus dem Benutzereingabefeld, dem Chat-Ausgabefeld, einer Liste

mit Benutzernamen, die andere teilnehmende Clients in demselben Raum zeigt, und eine Aktionsleiste. Mit letzterer können Sie einige Aktionen durchführen, wie etwa Änderungen des Benutzernamens, Verbindungsaufbau, Teile, Verbindungsabbau usw. Diese Aktivitäten können alle über denselben Prozess verwaltet werden, der das Ergebnis in einem Frame ausgibt.

Der Hauptprozess, der auch für den Datenstrom der Chat-Ausgabe verantwortlich ist, enthält Statusinformationen, die in einer Datenbank aktualisiert werden. Auf diese können alle anderen Schnittstellenelemente zugreifen, um Daten auszulesen und in geeigneter Weise anzuzeigen (siehe Abbildung 3.8).

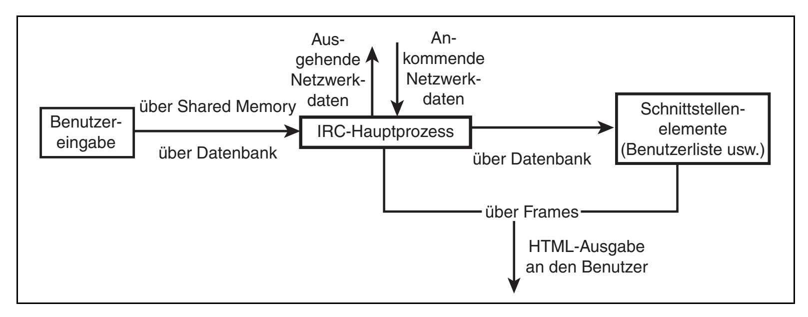

Abbildung 3.8: Das endgültige Programmlayout

3.5.5 Schnittstelle zum Entwickler

Eine Schnittstelle für Entwickler? Was hat das mit dem Chat zu tun? Und wie soll das funktionieren? Die meisten Programme sind »unflexibel«. Sie lassen sich gar nicht oder nur schwer durch fremde Entwickler verändern. Bei endbenutzer-orientierten Programmen (zum Beispiel Desktop-Umgebungen, wie etwa Windows, KDE, MacOS usw.) wird wohl kaum jemand die Ideallösung finden. Ähnlich wie bei einem Chat-System werden die meisten Leute, die das Programm herunterladen, bemerken: »Hey super, aber es fehlt dies oder das« oder »cool, aber ich mag nicht, wie es xyz tut«.

Ohne einen einfachen, klar ausgewiesenen Weg, wie das Programm durch den Benutzer verändert werden kann, werden die meisten Anwendungen im Müll landen. Sofern die Funktionsweise sie nicht sofort vom Hocker reißt, werden die meisten Leute nicht einmal versuchen, an einem Programm herumbasteln, das sie nicht selbst entwickelt haben.

Für die Chat-Programme bedeutet dies, dass wir einen festen Programmkern entwickeln müssen (den Teil der Anwendung, den niemand jemals ändern muss oder soll), der Schnittstellen zu einer Reihe von Plugins besitzt (dem Teil der Anwendung, den die meisten irgendwie ändern möchten). Damit erreichen wir eine Unabhängigkeit von Programmcode und Schnittstellenlayout (indem wir eine Schnittstelle für HTML-Entwickler bereitstellen) sowie eine Unabhängigkeit der einzelnen datenverarbeitenden Stufen voneinander).

Denken Sie noch einmal über die Wichtigkeit dieser Forderung nach. Möchten Sie, dass ein Programm so konzipiert ist? Brauchen Sie dies überhaupt? Überlegen Sie sich, wie Sie dies realisieren können.

3.5.6 Schnittstelle zum HTML-Entwickler

Im Falle der HTML-Schnittstelle wird die Trennung von Programmcode und Layout über Schablonen erreicht. Dies ist nicht nur der einfachste Weg, um ein Programm an Ihre Bedürfnisse anzupassen, sondern auch der leistungsfähigste. Innerhalb von Sekunden können Sie die Struktur und das Aussehen verändern, ohne eine einzige Programmzeile ändern zu müssen. Jeder Benutzer, der über grundlegende HTML-Kenntnissen verfügt, kann die Art und Weise ändern, wie sich ein Programm einem Benutzer präsentiert. Da wir dieses Verfahren an anderer Stelle im Buch ausführlich besprechen, möchten wir hier nicht näher darauf eingehen. Wenn Sie weitere Einzelheiten über die Verwendung von Schablonen erfahren wollen, lesen Sie Kapitel 5 »Grundlegende Webanwendungsstrategien«.

3.5.7 Schnittstelle zum Programmcodeentwickler

Die Bereitstellung einer Schnittstelle für andere Entwickler ist in der Regel mit dem Begriff »API« (Application Programming Interface) verknüpft. Eine API, also eine Anwendungsprogrammschnittstelle, wird normalerweise von Bibliotheken (wie phpIRC) bereitgestellt, aber nicht von vollständigen Programmen. Programme, die sich durch einen Programmierer erweitern lassen, sind jedoch sehr viel erfolgreicher, als Programme, die »wie vorliegend« verwendet werden müssen. Im Fall von PHP-Anwendungen kann natürlich jeder den Quellcode ändern, einige Leute schrecken jedoch davor zurück, ein komplexes System zu analysieren und Änderungen darin durchzuführen. Daher muss das Programm selbst einige Möglichkeiten zur Erweiterung aufzeigen.

Hinweis: Wir unterscheiden hier zwischen »Anwendung« und »Bibliothek«. Unter »Bibliothek« verstehen wir Programme, die nicht eigenständig ausgeführt werden können und sich in der Regel leichter erweitern lassen als Anwendungen. Anwendungen bestehen aus einem vollständig geschlossenen System.

Finden Sie heraus, wie Sie herkömmliche Programme erweitern können. Prüfen Sie beispielsweise bei Ihrem bevorzugten Textverarbeitungswerkzeug, ob der Entwickler eine Möglichkeit vorgesehen hat, die Funktionalität des Programms zu erweitern.

Im Laufe der Zeit haben sich hauptsächlich zwei Verfahren zur Programmerweiterung herausgebildet. Entweder bietet das Programm die Möglichkeit, Skripte zu erstellen (ähnlich wie Makros), oder das Programm kann Plugins verwenden. Bei PHP bedeutet die Implementierung einer Skriptsprache in einem zeitkritischen Teil eines Systems ... wir brauchen das gar nicht zu Ende zu denken. Darüber hinaus wäre die Entwicklung eines ausgereiften Parsers wirklich zu viel verlangt. Plugins lassen sich dagegen sehr viel leichter installieren und bieten viele Vorteile. Ein *Plugin* ist ein kleines Programmfragment, das sich selbst beim Programm anmelden kann und bestimmte Ereignisse aus diesem abfangen, auf interne Daten zugreifen kann usw. Es fügt sich zwar nahtlos in das Hauptsystem ein, bleibt aber trotzdem eine eigenständige Datei, die abgetrennt und separat verteilt werden kann. Es kann an das System angehängt werden, ohne dass Sie dafür eine Programmzeile ändern müssen. Damit kann auch ein Systemadministrator, der keine Kenntnisse in PHP besitzt, das Programm mithilfe eines fremden Programmcodes erweitern. Wie dies genau geht, zeigt Abbildung 3.9.

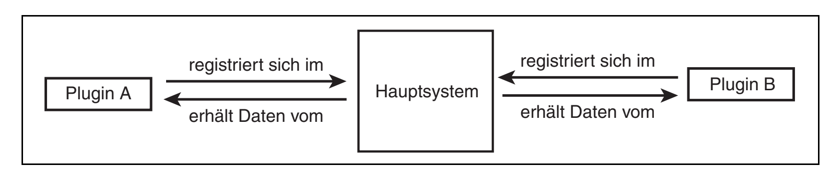

Abbildung 3.9: Chat-System mit Plugins

Entwerfen Sie, zumindest theoretisch, Ihr eigenes Plugin-System. Schreiben Sie ein minimales Programm, das Plugins registrieren und ausführen kann.

Beim Hochfahren fügt phpChat eine Include-Datei ein, die ihrerseits alle gewünschten Plugins integriert. Listing 3.3 zeigt, wie diese Include-Datei funktioniert:

```
////////////////////////////////////////////////////////////////////////////
//
// Plug-in Integrator
//
////////////////////////////////////////////////////////////////////////////

include("chat_plugin_out_htmlspecialchars.php3");

include("chat_plugin_out_link_transform.php3");

include("chat_plugin_out_colorcodes.php3");

include("chat_plugin_clock.php3");
```

```
include("chat_plugin_cmd_basic.php3");
include("chat_plugin_out_basic.php3");

/////////////////////////////////////////////////////////////////////////
```

Listing 3.3: Die Include-Datei als Plugin

Alle Plugins sind auf dieselbe Weise aufgebaut. Sie bestehen aus einem Hauptteil und einem Ereignisteil. Der Hauptteil ruft in PHP zwei Funktionen mit den folgenden Namen auf: `chat_register_plugin_init()` und `chat_register_plugin_deinit()`. Jede der Funktionen akzeptiert als Parameter den Namen einer weiteren Funktion, die für die Initialisierung bzw. Deinitialisierung aufgerufen werden sollen.

phpChat fügt diese Funktionsnamen in eine interne Tabelle ein. Bei der Initialisierung des Chat durchläuft phpChat, sobald es vollständig eingerichtet ist, die Initialisierungstabelle und ruft die Initialisierungsfunktion jedes Plugin auf, das sich selbst angemeldet hat. Beim Verbindungsabbau durchläuft es entsprechend die Deinitialisierungstabelle. Mit diesem Verfahren wird den Plugins signalisiert, dass sie sich aktivieren bzw. deaktivieren sollen.

Um im Programm nützlich zu sein, bietet phpChat eine Reihe von Ereignissen, an die sich jedes Plugin anhängen kann. Während der Initialisierung fordert jedes Plugin PHP auf, ihm eine Reihe von gewünschten Ereignissen zu schicken. Ein Ereignis kann der Leerlauf des Chats, die Weiterleitung einer neuen Nachricht des Benutzers, das Klicken auf einen Benutzernamen in der Liste der Benutzernamen oder eine vom Netzwerk ankommende Nachricht sein und vieles mehr.

Zur Laufzeit können die Plugins diese Ereignisse abfangen und bestimmte Aufgaben ausführen. Das Uhrzeit-Plugin meldet sich beispielsweise beim »Leerlauf«-Ereignis an und überprüft in bestimmten Abständen die Systemzeit. Nach einer vordefinierten Anzahl von Minuten teilt es dem Benutzer die Zeit mit.

Für die meisten Ereignisse sendet phpChat auch Parameter (wie etwa den Nachrichtentext bei ankommenden Nachrichten). Diese können von den Plugins geändert werden. Zu den in Listing 3.3 aufgeführten Plugins gehören beispielsweise die Plugins `htmlspecialchars` und `link_transform`. Sie ändern die Ausgabe der Nachrichten. `htmlspecialchars` ruft für alle gedruckten Texte (aus Sicherheitsgründen, so dass niemand einen böswilligen HTML-Code in den Chat einfügen kann) `htmlspecialchars()` auf. Der Verbindungstransformator erfasst alle URLs und Email-Adressen und stellt ihnen ein `<a href=""></a>` bzw. `mailto:` voran, so dass der Benutzer im Chat-Fenster die Verbindung direkt anklicken kann (siehe Abbildung 3.10).

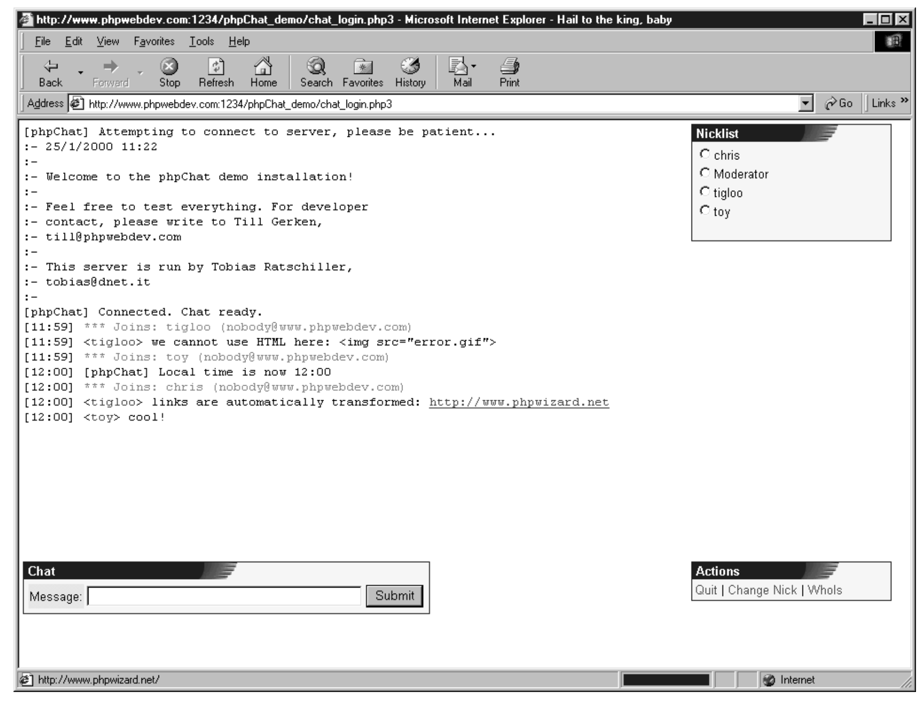

Abbildung 3.10: Plugins bei der Arbeit

Wie Sie sehen können, bieten Plugins eine extrem leistungsfähige Alternative, um ein komplexes System zu erweitern. Deshalb hat phpChat die meisten seiner internen Routinen als separate Plugins aus dem Programm ausgelagert. Der komplette Befehlsinterpreter wurde zu einem Plugin, genau wie alle Textformatierungs-/Textausgabeprozeduren. Damit bleibt nur noch ein harter Kern, der nicht mehr geändert werden muss, da er nichts enthält, was eine Änderung benötigt. Der Rest kann unbegrenzt verändert, erweitert oder gar entfernt werden, ohne dass dies Auswirkungen auf die Systemleistung oder die Funktionsfähigkeit hat. Haben Sie jemals erlebt, dass sich ein Programm nicht beschwert, wenn jemand seine Dateien löscht? Mithilfe dieser Technik wird sich das Programm nicht beschweren – und sich sogar dynamisch anpassen.

Plugins können vielfältig eingesetzt werden, nicht nur für Chat-Programme. Sie könnten beispielsweise ein Portal errichten, das aus der traditionellen Nachrichtenseite, einer Email-Schnittstelle usw. besteht. Mithilfe von Plugins können Sie einen »Website-Kern« konzipieren, der sich um alle grundlegenden Dinge kümmert, wie etwa die Bereitstellung eines Seitenlayouts, eines Datenbank-Backends, Sitzungsverwaltung usw. Basierend auf dem Website-Kern können Sie dann Plugins für die Anzeige von Nachrichten, Versand und Empfang von Emails und selbst die Bereitstellung verschiedener Anmelde-

verfahren erzeugen. Auch wenn dies mit einigem Aufwand verbunden ist, möchten wir Sie ermuntern, als Übung eine plugin-basierte Anwendung zu erstellen. Es wird der Mühe wert sein.

Listing 3.4 zeigt eine Plugin-Schablone, die einen »Dummy«-Plugin als Codebasis für neue Plugins implementiert.

```
<?

//
// Use these variables to tell the plug-in installer how you named your
// initialization and deinitialization functions. This is done to eliminate
// the need for changing the installer code, which would ask for errors.
//
$plugin_init_function = "myplugin_init";
$plugin_deinit_function = "myplugin_deinit";
//
//
//

///////////////////////////////////////////////////////////////////////////
//
// myplugin_idle_callback(int code, mixed parameter) - example callback
//
///////////////////////////////////////////////////////////////////////////
//
// This is an example for a callback function. See below on how to register
// and remove it from the call chain.
//
// $code specifies the reason for invocation, $parameter contains all
// callback information.
//
// The return value should always consist of a modified or unmodified
// version of the input parameter $parameter. The return value is used as
// input parameter for the next callback. This allows for multi-stage
// message processing and such.
//
///////////////////////////////////////////////////////////////////////////

function myplugin_idle_callback($code, $parameter)
{

    return($parameter);

}

///////////////////////////////////////////////////////////////////////////
//
// myplugin_init() - initializes this plug-in
```

```
//
/////////////////////////////////////////////////////////////////////////////
//
// Put all your initialization code in here. This code will be called as
// soon as the main bot is all set up with connecting and callback
// installation; thus, you can rely on a safe environment.
//
// Although the return value is currently not used, "0" should indicate
// initialization failure and "1" initialization success. This might be
// used later on to enable plug-ins to stop the current chat session
// right after login.
//
/////////////////////////////////////////////////////////////////////////////
//
// Return value:
// 0 - error
// 1 - success
//
/////////////////////////////////////////////////////////////////////////////

function myplugin_init()
{

    // register callbacks here
    chat_register_callback(CHATCB_IDLE, "myplugin_idle_callback");

    return(1);

}

/////////////////////////////////////////////////////////////////////////////
//
// myplugin_deinit() - deinitializes this plug-in
//
/////////////////////////////////////////////////////////////////////////////
//
// All deinitialization code should go here. This function is called before
// the bot goes down; thus, all network connections are still active.
//
// Although the return value is currently not used, "0" should indicate
// deinitialization failure and "1" deinitialization success. This might be
// used later on to force delayed shutdowns.
//
/////////////////////////////////////////////////////////////////////////////

function myplugin_deinit()
{
```

```
    // remove callbacks here
    chat_remove_callback(CHATCB_IDLE, "myplugin_idle_callback");

    return(1);

}

/////////////////////////////////////////////////////////////////////////////
//
// NOTE: DO NOT CHANGE ANYTHING BELOW THIS POINT!
//
/////////////////////////////////////////////////////////////////////////////

// installer code starts here

// register initialization function
chat_register_plugin_init($plugin_init_function);

// register deinitialization function
chat_register_plugin_deinit($plugin_deinit_function);

// installer code done

/////////////////////////////////////////////////////////////////////////////

?>
```

Listing 3.4: Eine Plugin-Schablone

Der Hauptcode registriert die Initialisierungs- und Deinitialisierungsroutinen für diesen Plugin. Wenn dies geschehen ist, installiert die Initialisierungsroutine die Rückrufe, die dieses Plugin abfangen möchte, und die Deinitialisierungsroutine entfernt sie.

3.6 Verwaltung und Sicherheit

Ein System ist nur dann ein gutes System, wenn es verwaltet werden kann. Die Tage, in denen es laut »Netiquette« eine Frage der Ehre war, sich selbst in die Gemeinschaft zu integrieren, sind längst vorbei. Heutzutage ist fast jeder Häckern, Schikanierungen und anderen Formen von Attacken ausgesetzt, und leider bleiben die meisten nicht auf der verbalen Ebene. Es gibt kaum etwas dagegen einzuwenden, nach Sicherheitslecks und anderen Löchern im Programm oder dem Netzwerksystem zu suchen. Diese auszunutzen, ist jedoch mehr als verwerflich. Trotzdem betrachten es viele als »Spaß«. Dieses Verhalten zeigt klar den Bedarf für eine externe Schnittstelle, die unabhängig

vom Hauptsystem arbeitet. Mit solch einer Schnittstelle erhalten Sie die vollständige Kontrolle über alle Daten und Benutzer des Programms. Für das Chat-Programm bedeutet dies, dass wir Benutzer hinauswerfen, ihre Nachrichten entschärfen und Chat-Räume sichern können.

Hinweis: Der Programmcode auf der CD-ROM enthält nicht alle hier aufgeführten Funktionen. Das Basisverwaltungssystem ist komplett und vollständig funktionsfähig. Aber wir möchten, dass Sie die Codebasis um die Funktionen erweitern, die Sie für passend halten. Wenn Sie bisher noch keine größeren Erweiterungen für umfangreiche Programme geschrieben haben, empfehlen wir Ihnen wärmsten, diese Erfahrung nun nachzuholen.

Die Frage für unser Chat-Programm lautet: Wo passen wir die Verwaltung ein? Folgende Möglichkeiten stehen uns zur Verfügung:

- Auf der Netzwerkebene: Wir könnten die Benutzer filtern, die eine Verbindung zur Server herstellen.
- Auf der PHP-Ebene: Wir könnten verhindern, dass sich Benutzer für den Chat anmelden.
- Auf der Datenbankebene: Wir könnten Nachrichten von Benutzern aus der Datenbank verwerfen.
- Auf der IRC-Ebene: Wir könnten die IRC-eigenen Netzwerkverwaltungsfunktionen verwenden.

3.6.1 Auf der Netzwerkebene

Die Sicherung auf Netzwerkebene lässt uns nur zwei Möglichkeiten: Entweder wir lassen eine Verbindung zu oder nicht. Hierfür können wir eine Firewall verwenden oder andere Möglichkeiten der IP-Maskierung nutzen. Diese Methode ist begrenzt, kompliziert, unsicher und im Allgemeinen nicht das, was wir wollen.

3.6.2 Auf der PHP/Webserver-Ebene

Bei der Sicherung auf Webserver-Ebene werden Verbindungen zum Server prinzipiell zugelassen, aber die Anmeldung des Clients wird durch einen Passwortschutz begrenzt (oder andere Berechtigungsverfahren). Genaugenommen läuft es wieder darauf hinaus, eine Verbindung zuzulassen oder nicht – was nicht wirklich befriedigend ist.

Dieses Verfahren kann jedoch zur Emulierung von Benutzerverboten verwendet werden. Die für IRC üblichen Verbote, nämlich K-Verbindungen und G-Verbindungen (lokale und globale Benutzerverbote), können bei einem

web-basierten Chatsystem nicht benutzt werden, da alle Verbindungen vom Webserver ausgehen. Die einzige Adresse, die man verbieten könnte, wäre die Adresse des Webservers, wodurch die komplette Schnittstelle vom Netz abgetrennt würde. Um trotzdem bestimmte Benutzer ausfiltern zu können, sollten die Verbindungen auf der PHP-Ebene ausgewertet werden.

3.6.3 Auf der Datenbankebene

Auf der Datenbankebene sieht das vollkommen anders aus. Clients können sich anmelden und chatten, aber ihre Nachrichten und Sitzungsdaten werden in der Datenbank gefiltert. Entweder durch ein externes Werkzeug oder den Chatcode selbst wird überprüft, ob der Benutzer das Recht hat, etwas zu sagen oder zu tun. Basierend auf dieser Information wird seine Nachricht in der Datenbank eingefügt oder nicht. Diese Strategie erfordert jedoch eine sehr enge Integration in den Hauptchat-Code. Außerdem ist sie nicht sehr flexibel (und etwas umständlich) und nicht sehr elegant zu implementieren.

3.6.4 Auf der IRC-Ebene

IRC hat eigene Verwaltungsfunktionen, die im Servercode und im Netzwerkprotokoll integriert sind (wir hoffen, Sie lesen die RFC und sind mit diesen Möglichkeiten vertraut). Die Verwaltung kann sogar durch reguläre Benutzer vorgenommen werden. Es gibt drei Ebenen:

- *Kanaloperatoren*. Diese Operatoren haben administrative Kontrolle über Kanäle. Sie können Benutzer hinauskicken, sie zum Schweigen bringen, sie verbieten, andere Benutzer zu Operatoren machen u.ä. (diese Ebene ist für alle Benutzer verfügbar).
- *IRC-Operatoren*. Diese Operatoren haben administrative Kontrolle über das Netzwerk (aber nicht über die Kanäle). Sie können Benutzer aus dem Netz werfen, sie verbieten, Netzwerkverbindungen herstellen usw. (diese Ebene ist nur bestimmten Benutzern zugänglich).
- *Dienste*. Dienste haben administrative Kontrolle über Kanäle, aber nicht über das Netzwerk. Sie können sich nicht wie reguläre Benutzer verhalten. Sie benötigen eine spezielle Anmeldeprozedur (diese Ebene ist nur für bestimmte Benutzer zugänglich und ist für automatisierte Clients gedacht).

Wie Sie sehen, kann die Verwaltung auf IRC-Ebene über einen Client durchgeführt werden, der separat vom Hauptchat-System arbeitet. Ein separater Client mit IRC-Operator- und Kanaloperator-Status ergäbe die ideale Kombination an Funktionen, die wir als Verwaltungssystem haben müssten. Im Prinzip brauchen wir zunächst nur den IRC-Operator-Status, da der Client, sobald er den IRC-Operator-Status erlangt hat, überall den Kanaloperator-

Status bekommen kann, indem er alle Benutzer in einem Kanal entfernt. Dies ist zwar keine sehr nette Methode, aber effektiver und vielseitiger, als den IRC-Servercode zu korrigieren, um IRC-Operatoren dieselben Rechte wie Kanaloperatoren zu verleihen.

3.7 Implementierung

Die Chat-Verwaltung sollte sich ähnlich wie das Hauptchat-Skript installieren lassen. Dazu wird ein Bot gestartet, der sich mithilfe von phpIRC im IRC anmeldet und versucht, sich als IRC-Operator zu registrieren. Danach wartet er auf weitere Befehle von der Web-Schnittstelle. Diese Befehle werden wie jene in einem entfernten datenbankbasierten Prozeduraufruf ausgegeben. Der Bot fragt in regelmäßigen Abständen eine Tabelle in der Datenbank ab, welche die Eingabebefehle für ihn enthält. Die Befehle werden von der Web-Schnittstelle in die Datenbank eingefügt. Sie bestehen aus einem Funktionsnamen, einer Sitzungskennnummer (ID) und einem Parameterfeld. Sobald der Bot einen neuen Befehl in der Datenbank findet, führt er diesen aus und schreibt die Ergebnisse des Befehls zusammen mit der Sitzungskennnummer in eine Ausgabetabelle. Die Web-Schnittstelle muss daher nur einen Befehl mit einer selbstgenerierten Sitzungskennnummer schreiben und warten, bis ein Datensatz mit derselben Sitzungskennnummer in der Ausgabetabelle auftaucht (siehe Abbildung 3.11).

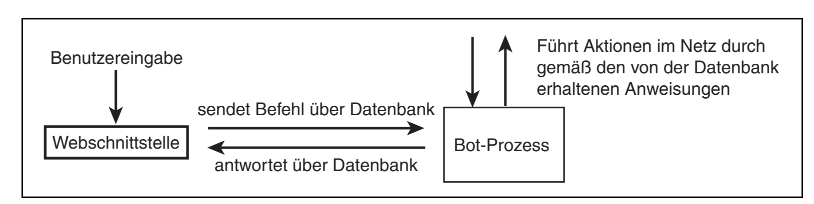

Abbildung 3.11: Datenbankbasierte RPC-Kontrolle über den Verwaltungs-Bot

Diese Methode ermöglicht eine flexible Fernkontrolle über den Bot.

3.8 Zusammenfassung

In diesem Kapitel haben Sie anhand eines Praxisbeispiels gelernt, wie Sie ein Entwicklungsprojekt planen. Wir sind auf die typischen Entwicklungsstadien eingegangen:

- Analyse der Anforderungen
- Wahl einer passenden Technologie
- Definition von Schnittstellen und APIs
- Implementierung

Sie sind uns durch den gesamten Entwicklungsprozess gefolgt, und wir haben aus unseren Beispielen Schlussfolgerungen gezogen, die für die meisten Software-Projekte zutreffen. Mit diesem Hintergrundwissen sind Sie für den nächsten Teil dieses Buches gerüstet, in dem wir Ihnen einige wichtige Konzepte für Webanwendungen zeigen.

Teil 2: Webanwendungen

4. Webanwendungskonzepte
5. Grundlegende Webanwendungsstrategien
6. Datenbankzugriff mit PHP
7. Anwendungen der Spitzentechnologie
8. Fallstudien

Pearson
Education

ADDISON-WESLEY

4 Webanwendungskonzepte

Wir bringen Speichen in einem Rad zusammen,
aber das innere Loch bringt den Wagen ins Rollen.
Wir formen Ton zu einem Topf,
aber erst die Leerheit in ihm hält den gewünschten Inhalt.
Wir bearbeiten Holz für ein Haus,
aber erst der Innenraum macht es bewohnbar.

Um die Bedeutung von Webanwendungskonzepten zu verstehen, müssen Sie zwischen Anwendungen und einzelnen Skripten unterscheiden. Ein Skript ist ein Hilfsprogramm und hat als solches keinen Kontext. Es kennt andere Skripte in Ihrem System nicht. Eine Anwendung ist dagegen für höhere Aufgaben ausgelegt. Es muss seinen Status überwachen und Transaktionen durchführen, weil es interaktiv ist. Da es in der Regel sehr viel mehr Benutzerinteraktionen verzeichnet als ein einzelnes Skript, müssen Sie sich auch über die Sicherheit und die Tauglichkeit Gedanken machen. Der Lohn für diese ganze Mühe ist, dass Sie den nächsten Yahoo! selbst generieren können. Durch Anwendungen wird das Web erst interessant.

4.1 HTTP und Sitzungen

Denken Sie an ein KDE-Programm, z.B. Kedit (KDE ist ein Fensterverwaltungssystem für Linux und kompatible X-Systeme). Ein typischer Vorgang könnte sein, eine Datei zu öffnen, den Inhalt zu ändern und die Datei unter einem anderen Namen wieder abzuspeichern. Kedit weiß zu jedem Zeitpunkt des Prozesses, was Sie tun. Es weiß, dass Sie die Datei bearbeiten, wo sich der Cursor befindet, wohin Sie die Maus bewegen usw. Selbst wenn Sie eine zweite Instanz von Kedit öffnen, wird ihn dies nicht irritieren. Wenn Sie in Instanz 1 »Save« wählen, wird nicht die Datei aus Instanz 2 gespeichert. Dies ist möglich, weil Kedit (oder das Betriebssystem, um genau zu sein) Ihre Aktionen einer bestimmten Instanz des Programms zuordnen kann – und zwar dadurch, dass es ein Ereignis erhält wie: »In der Instanz mit der PID 4711 (eine PID ist ein eindeutiger Prozessbezeichner in UNIX-Systemen) wurde die Maus zu den Koordinaten 10, 4 bewegt«.

4.1.1 Statuskontrolle

Als Tim Berners-Lee 1991 das Hypertext-Transfer-Protocol entwarf, beschloss er, HTTP so schnell wie möglich zu machen und daher alle Statusinformationen wegzulassen. [1]

HTTP besitzt keinen Mechanismus, um den Status zu speichern. Deshalb spricht man bei HTTP von einem kontextfreien oder zustandslosen Protokoll. Die einzelnen Anfragen werden nicht zueinander in Beziehung gesetzt. Der Webserver (und damit auch PHP) kann einzelne Benutzer nicht einfach voneinander unterscheiden und kennt keine Benutzersitzungen. Deshalb müssen wir einen eigenen Weg finden, um einen Benutzer zu kennzeichnen und ihm Sitzungsdaten zuzuordnen (d.h. alle Daten, die Sie für einen Benutzer speichern möchten). Wir benutzen den Begriff Sitzung für eine Instanz eines Benutzers, der eine Website und eine oder mehrere Seiten anzeigt. Eine typische Online-Einkaufssitzung könnte beispielsweise daraus bestehen, einen Artikel in den Einkaufswagen zu legen, auf die Abmeldeseite zu gehen, die Adresse und die Kreditkartendaten einzugeben, den Auftrag abzuschicken und das Fenster zu schließen.

Die gute Nachricht ist, dass es mehr als eine Möglichkeit gibt, Sitzungen zu verwalten. Die schlechte Nachricht ist, dass keine von ihnen perfekt ist. Lassen Sie uns zunächst jene aussortieren, die nicht funktionieren, selbst wenn sie auf den ersten Blick als gute Alternativen erscheinen.

Der typische PHP-Programmierer wird zunächst versuchen, das Problem zu ignorieren und einen Umweg zu finden. Die offensichtlichste Methode ist, alle Daten anstatt auf dem Server auf dem Client zu speichern. Dies führt zu Formularen mit sehr vielen versteckten Feldern oder sehr langen URLs. Dies wird schnell unpraktisch, wenn mehr als zwei Dateien und mehr als eine Variable gespeichert werden müssen. Ein etwas besserer Ansatz ist die Verwendung von Cookies, mit denen alle Daten auf der Client-Seite gespeichert werden können.

Es gibt jedoch verschiedene Gründe, warum die Daten nicht auf der Client-Seite gespeichert werden sollten:

- Sie verlieren die Kontrolle über die Daten. Solange der Benutzer nicht auf Ihre Website zurückkehrt, haben Sie keinen Zugriff auf die Daten. Viel schlimmer ist, dass die Daten möglicherweise manipuliert sind, wenn Sie sie zurückerhalten. Neunzig Prozent aller Website-Abstürze sind durch Programme verursacht, die verfälschte Daten von der Client-Seite akzeptieren und diesen Daten vertrauen. Bewahren Sie keine Daten auf dem Client auf. Vertrauen Sie keinen Daten vom Client.
- Wenn Sie `GET/POST` verwenden, bleiben die Daten nicht bis zur nächsten Sitzung erhalten.
- Wenn Sie sich ausschließlich auf Cookies verlassen, haben Sie das Problem, dass nicht alle Benutzer Cookies akzeptieren. Sie deaktivieren die Cookies einfach auf ihrem Browser.
- Der Datenerhalt ist sehr schwierig, weil Sie alle Daten auf jeder Seite speichern müssen. Jede Variable muss URL-kodiert als verstecktes Feld in einem Formular hinzugefügt, an die URL angehängt oder als Cookie

gespeichert werden. Dies ist bereits bei einer einzelnen Variablen, wie etwa der Sitzungskennnummer, sehr schwierig, ganz zu schweigen bei Dutzenden von Variablen!

Daher sollten Sie die Daten auf dem Server speichern. Wo Sie diese genau ablegen, ist nicht wichtig; es kann in einem relationalen Datenbank-Verwaltungssystem (RDBMS), einer reinen Textdatei, einer dBASE-Datei usw. sein. Vorzugsweise sollten Sie eine relationale Datenbank als Speichermedium wählen, da Webanwendungen in der Regel bereits solche Datenbanken wie etwa MySQL verwenden.

Um die Daten einem Benutzer zuzuordnen, brauchen Sie eine Sitzungskennnummer, einen Schlüssel, der einen Benutzer an seine Daten bindet. Wie jedoch zuvor erwähnt, hat HTTP jedoch keinen Mechanismus, um Benutzer zu kennzeichnen. Wie sollten Sie den Benutzer dann kennzeichnen?

Eine Idee, die einem sofort in den Sinn kommt, ist die Verwendung der Benutzer-IP-Adresse. Dieser Ansatz mag zwar auf den ersten Blick logisch erscheinen, er scheitert jedoch an den mit ihm verbundenen Problemen:

- Viele Service Provider zwingen Benutzer einer Wählleitung Proxyserver zu verwenden. `$REMOTE_ADDR` zeigt hier die IP des Proxy. Wenn zwei AOL-Benutzer versuchten, die Webanwendung zur gleichen Zeit zu verwenden, gäbe es ein Durcheinander.
- Einige Service Provider (z.B. Provider für den Kabelzugriff) ändern die IP-Adressen ihrer Benutzer von Zeit zu Zeit, um zu verhindern, dass sie von Webservern ausgeführt werden.
- Schließlich könnte es passieren, dass ein Benutzer seine Internetverbindung schließt, einen Kaffee trinken geht, um 15 Minuten später wieder im Online-Geschäft zu erscheinen (natürlich mit einer anderen IP).

Nachdem Sie die Tatsache akzeptiert haben, dass es keinen allgemeinen Ansatz gibt, um den Benutzer mit einer vordefinierten magischen Nummer zu kennzeichnen, bleibt Ihnen nur noch die Möglichkeit, selbst eine Sitzungskennnummer (Session-ID) zu erzeugen und diese von Seite zu Seite weiterzuleiten. (»Wie?« werden Sie fragen. Lesen Sie weiter, wir liefern die Einzelheiten ein wenig später.) Die ID muss willkürlich sein, da ansonsten ihre Benutzer versuchen werden sie herauszufinden, um andere Sitzungen zu übernehmen. Wenn die ID linear ist, z.B. eine normale Nummer (`page.php3?ID=5`), können Sie darauf wetten, dass mindestens ein Benutzer versuchen wird `page.php3?ID=6` zu öffnen. Vielleicht ist es Ihnen einfach nur peinlich, wenn normale Benutzer die Einkaufswagen anderer Benutzer einsehen können. Es wird jedoch zu einem gefährlichen Sicherheitsproblem, wenn Hacker Sitzungen von Anderen übernehmen, um die Kreditkartennummer herauszubekommen oder gefälschte Aufträge zu erzeugen.

PHP hat eine eingebaute uniqid()-Funktion. Diese basiert jedoch auf der Systemzeit und ist nicht sicher genug, um als Sitzungskennnummer zu dienen. Sie können sie jedoch mit einer Hash-Funktion und rand() kombinieren, um eine wirklich willkürliche Zeichenfolge mit 2^{128} möglichen Elementen zu konstruieren:

```
srand((double)microtime()*1000000); // Seed the random number generator
$session_id = md5(uniqid(rand())); // Construct the session ID
```

Zugriff auf die IP-Adresse des Benutzers

Sie können auf die IP-Adresse des Benutzers über die Umgebungsvariable $REMOTE_ADDR zugreifen. Mit phpinfo() erhalten Sie eine Liste aller verfügbaren Umgebungsvariablen.

Jeder, der versuchte, dies zu knacken, müsste im Brechstangenverfahren alle möglichen Elemente attackieren, um eine gültige Sitzungskennnummer aus 340.282.366.920.938.463.463.374.607.431.768.211.456 möglichen Werten herauszufinden. Die Kryptoanalytiker van Oorschot und Wiener entwickelten eine theoretische Suchmaschine für MD5 und schätzen 1994, dass eine solche Maschine (geschätzte Kosten $ 10 Millionen) im Durchschnitt 24 Tage brauchen würde, um eine mit MD5 verschlüsselte Nachricht zu knacken.[2]

Wenn Sie dies beunruhigt, sollten Sie darüber nachdenken, Ihren Server vom Internet abzuklemmen.

md5(uniqid()) – dasselbe Gebilde wie oben, ohne rand()-Aufruf – wäre übrigens nicht willkürlich genug. Da uniqid() auf der Systemzeit basiert, lässt sie sich erraten, sobald der Hacker die Systemzeit des Servers kennt. Der zu durchsuchende Bereich ist dann deutlich kleiner als 2^{128}.

4.1.2 Sitzungskennnummer-Verteilung mit Cookies

Die einzige Aufgabe, die noch verbleibt, ist, die Sitzungskennnummer für alle Seiten Ihres Programms verfügbar zu machen. Dies erreichen Sie beispielsweise, indem Sie einen Cookie setzen, der die ID enthält. Wenn Sie einen Benutzer über mehrere Sitzungen hinweg identifizieren wollen, ist die Verwendung eines Cookies sogar die einzige Möglichkeit. Unglücklicherweise wird ein Teil der Benutzer den Cookie in ihrem Browser deaktiviert haben (Schätzungen zufolge sind es bis zu 20%). Je nach Zielgruppe lohnt es sich möglicherweise, die Benutzer auf eine Hilfsseite umzuleiten, die ihnen zeigt, wie Sie die Cookies aktivieren können.

Die Sitzungskennnummer mit Cookies zu übergeben, ist für den Entwickler die bei weitem einfachste Methode. Ihr Programm muss nichts anderes tun, als das Cookie zu setzen.

4.1.3 Manuelle Änderung der URL

Eine weitere Alternative, um die Sitzungskennnummer zu verbreiten, ist, die URL manuell zu ändern. Damit leiten Sie die Sitzungskennnummer entweder über GET/POST weiter oder »verstecken« sie im URL. Dazu müssen Sie die HTML-Tags frame, form und a ändern, um auf Ihre ID zu referenzieren:

```
// A frame source definition
printf('<frame src="page.php3?session_id=%s">', $session_id);

// A hidden form field
printf('<input type="hidden" name="session_id" value="%s">', $session_
id);

// A normal link
printf('<a href="page.php3?session_id=%s">Link</a>', $session_id);
```

Wenn Ihr Programm Abbildungen, eingebettete Frames oder JavaScript-Umleitungen enthält, müssen Sie auch diese ändern.

Die Änderung der URL hat mehrere Nachteile:

- Sie bedeutet einen beträchtlichen Mehraufwand für Sie als Entwickler. Sie müssen die Sitzungskennnummer für alle Verknüpfungen manuell hinzufügen. Wenn Sie nur eine einzige Verknüpfung vergessen haben, geht die Sitzung des Benutzers verloren.
- Es zeigt, dass Ihre Seiten dynamisch erstellt werden. Einige Suchmaschinen werden sich weigern, die Seiten überhaupt zu indizieren. Andere schneiden alles nach dem Fragezeichen von der URL ab.
- Die Sitzungskennnummer wird in den Lesezeichen und Ausdrucken des Benutzers hinzugefügt. Wir kennen sogar Artikel in technischen Fachzeitschriften, in denen die Sitzungskennnummer einer Website als Teil einer Referenz angegeben wurde. Vom Standpunkt der Benutzerfreundlichkeit lässt sich sagen, dass es für Benutzer schwieriger ist, die URL manuell zu ändern, um bestimmte Ressourcen auf einer Website zu finden.
- Die Sitzungskennnummer wird in den Proxy-Servern protokolliert und ist in der CGI-Umgebungsvariablen HTTP_REFERER für andere Websites sichtbar.

4.1.4 Dynamische Pfade

Lassen Sie uns sehen, ob wir einige der Nachteile vermeiden können, die sich bei der Neuschreibung der URL ergeben. Zunächst können Sie die ID Ihrer URL in der Amazon.com-Weise hinzufügen (siehe Abbildung 4.1). Diese sehen dann so aus: http://server.com/page.php3/<session-id>. Auf diese Weise ist die Sitzungskennnummer ein Teil des Pfads zum Skript, und für Suchmaschinen und Spider wirkt die URL wie eine statische Seite. Dies funktioniert, weil der Webserver weiß, dass page.php3 ein Skript ist und in der URL nicht

nach weiteren Dateien sucht. Damit ist die Sitzungskennnummer aber nicht mehr automatisch in Ihrem PHP-Skript verfügbar. Um auf sie zuzugreifen, müssen Sie den Pfad selbst analysieren:

```
function session_start_from_path()
{
    global $HTTP_HOST, $REQUEST_URI;

    ereg("/([0-9a-z]{32})", $REQUEST_URI, $regs);
    $session_id = $regs[1];

    if(!isset($session_id) || empty($session_id))
    {
        srand((double)microtime()*1000000);
        $session_id = md5(uniqid(rand()));

        $destination = "http://$HTTP_HOST$REQUEST_URI/$session_id";
        header("Location: $destination");
    }

    session_id($session_id);
    session_start();
}
```

Abbildung 4.1: Amazon.com versteckt die Sitzungskennnummer in der URL

Alle anderen Nachteile der manuellen URL-Änderung bleiben jedoch bei dynamischen Pfaden bestehen.

Dynamische Pfade mit mod_rewrite

Mit einen kleinen Trick können Sie sich zumindest die ärgerliche manuelle Codierung der Sitzungskennnummer ersparen. Wie wäre es, wenn die URL folgendermaßen aussähe?

```
http://server.com/<session-id>/page.php3
```

Hier betrachtet der Browser die Sitzungskennnummer als Teil des Verzeichnisses und würde sie bei jeder Anfrage verschicken. Wenn Sie dieses Format unverändert verwenden, würden Sie allerdings nur den Fehler »File not found« erhalten, da es kein Verzeichnis gibt, das wie die Sitzungskennnummer aussieht. Wir brauchen eine Methode die Sitzungskennnummer aus dem Pfad zu entfernen, bevor der Webserver die URL tatsächlich sieht.

Hier kommt `mod_rewrite` ins Spiel. `mod_rewrite` ist ein Apache-Modul, das komplexe Umwandlungen von regulären Ausdrücken in eine URL durchführt, bevor es diese an den Apache-Server weiterleitet. Mit `mod_rewrite` können wir die Sitzungskennnummer einfach aus der URL streichen. Dies ist eine interne Änderung der URL, die nur Apache sehen wird, nicht aber der Client. Apache sieht eine normale Anfrage ohne Sitzungskennnummer, die aber noch in den üblichen Variablen für PHP verfügbar ist.

Einlesen von mod_rewrite

Das Modul `mod_rewrite` wird nicht standardmäßig in Apache kompiliert. Informieren Sie sich in der INSTALL-Datei von Apache über die Anweisungen, mit denen Sie Apache mit diesem Modul kompilieren können.

Ein URL, wie der folgende

```
http://www.server.com/b6ac8ca8e453cdc43e6078abf044cdb5/script.php3
```

lässt sich mit dieser Umschreibungsregel ändern

```
RewriteEngine On
RewriteBase /
RewriteRule ^[0-9a-z]{32}/(.+) /$1
```

Die erste Zeile weist `mod_rewrite` an zu starten. Die zweite Zeile setzt explizit ein Basisverzeichnis, das wir nur brauchen, wenn wir `mod_rewrite` in einem lokalen Kontext verwenden, z.B. in einer *.htaccess*-Datei. Die dritte Zeile definiert schließlich den regulären Ausdruck, der für die URL-Neuschreibung verwendet wird. Unser Ausdruck wirft die Sitzungskennnummer einfach aus der URL raus.

Um die Sitzung zu starten, verwenden wir eine ähnliche Funktion, wie jene, die wir bereits geschrieben haben. Nur die anfängliche Umleitung ist anders.

```
function session_start_from_rewrite()
{
    global $HTTP_HOST, $REQUEST_URI;

    ereg("/([0-9a-z]{32})", $REQUEST_URI, $regs);
    $session_id = $regs[1];

    if(!isset($session_id) || empty($session_id))
    {
        srand((double)microtime()*1000000);
        $session_id = md5(uniqid(rand()));

        $destination = "http://$HTTP_HOST/$session_id$REQUEST_URI";
        header("Location: $destination");
    }

    session_id($session_id);
    session_start();
}
```

Bei allen Anfragen geht der Browser davon aus, dass die Sitzungskennnummer in der URL enthalten ist; er wird daher bei jeder Anfrage die Sitzungskennnummer automatisch senden. Sie ersparen sich damit die Mühe, alle Verknüpfungen selbst zu ändern, jedoch nur, wenn Sie relative Verknüpfungen in Ihrem Programm verwenden. Wenn Sie absolute URL-Pfade verwenden (z.B. */directory/script.php3*), müssen Sie diese trotzdem manuell ändern.

Dynamische Pfade mit PHP 4.0

Die automatische Änderung des URL ist eine der wirklich phantastischen neuen Funktionen von PHP 4.0. Um sie zu aktivieren, müssen Sie PHP mit `--enable-trans-id` neu konfigurieren und es erneut kompilieren. Dann wird die Sitzungskennnummer in der Form `<session-name>=<session-id>` automatisch in allen relativen Verknüpfungen auf Ihren mit PHP analysierten Seiten eingefügt.

Nicht für Hochleistungs-Websites

Auch wenn es eine nette Funktion ist, sollten Sie diese nur mit Vorsicht in Hochleistungs-Websites einsetzen. PHP muss auf jeder einzelnen Seite nachsehen, diese analysieren, um festzustellen, ob sie relative Links enthält, und schließlich die ID den Links hinzufügen. Dies hat natürlich Auswirkungen auf die Leistung. Wir empfehlen statt dessen `mod_rewrite` oder DNS-Tricks zu verwenden.

Auf die Verbreitung der Sitzungskennnummer in realen Anwendungen gehen wir später ein. Zunächst möchten wir Ihnen eine weitere Methode der Sitzungskennnummer-Verbreitung vorstellen, die wohl die spektakulärste ist.

4.1.5 DNS-Tricks

Die Notwendigkeit, alle Links in einem Programm mit der Sitzungskennnummer auszuzeichnen, kann wirklich lästig sein. Bei PHP 4.0 gibt es eine Möglichkeit, dies automatisch durchzuführen. Bei größeren Websites kann das zu erheblichen Leistungseinbußen führen, und diese Methode funktioniert nicht mit PHP 3.0.

Möglicherweise haben wir eine Lösung für Sie.

Zunächst die Voraussetzungen: Sie müssen in der Lage sein, den DNS-Datensatz für Ihren Server zu ändern, und der Server, den Sie für diese Art der Sitzungskennnummer-Verbreitung verwenden möchten, braucht eine eigene statische IP. Virtuelle namensbasierte Hostrechner funktionieren hier nicht.

Sie können diese Anforderungen erfüllen? Großartig! Wenn Sie sich mit Namensservern auskennen, wissen Sie vielleicht, dass Sie bei der DNS-Konfiguration Wildcard-Einträge machen können. Diese Einträge bilden in der Regel einen willkürlichen Hostnamen auf eine bestimmte IP ab, z.B. folgenden Eintrag, der Anfragen für alles unterhalb von `phpwebdev.com` an die IP 194.242.199.228 umleitet:

```
*.phpwebdev.com    IN    A    194.242.199.228
```

Eine Anfrage nach `http://this.is.one.cool.domain.phpwebdev.com` wird zu der angegebenen IP umgeleitet. Da der Hostname willkürlich ist, muss der Apache-Server so konfiguriert werden, dass er mit der IP arbeitet – im Gegensatz zu namensbasierten virtuellen Hostverfahren, bei denen der Hostname fest und bekannt sein muss. Unserer Apache-Konfiguration sieht folgendermaßen aus:

```
<VirtualHost 194.242.199.228>
    ServerAdmin tobias@dev.phpwebdev.com
    DocumentRoot /home/www/htdocs
    ServerName phpwebdev.com
</VirtualHost>
```

Unser Trick funktioniert auch, wenn der Apache-Hauptserver dieser Adresse zugeordnet ist.

Das Geheimnis besteht darin, die Sitzungskennnummer im Hostnamen selbst zu verschlüsseln. Bei der ersten Anfrage an das Programm wird die Sitzungskennnummer erstellt, und der Client wird an die neue URL umgeleitet, die den gekennzeichneten Hostnamen enthält. Dies sieht folgendermaßen aus:

```
355e1bce8828d4fb5c83c1e35ad02caa.phpwebdev.com
```

Der Vorteil liegt klar auf der Hand: Solange Sie in Ihrem Programm relative Links verwenden, brauchen Sie sich nicht länger mit der manuellen Änderung von URLs herumzuschlagen!

Wir haben die bereits erwähnte Funktion zum Sitzungsstart so modifiziert, dass sie die Sitzungskennnummer aus dem Hostnamen extrahiert:

```
function session_start_from_host($host)
{
    global $HTTP_HOST, $PHP_SELF;

    ereg("([0-9a-z]{32})\.", $HTTP_HOST, $regs);
    $session_id = $regs[1];

    if(!isset($session_id) || empty($session_id))
    {
        srand((double)microtime()*1000000);
        $session_id = md5(uniqid(rand()));

        $destination = "http://$session_id.$host$PHP_SELF";
        header("Location: $destination");
    }

    session_id($session_id);
    session_start();
}
```

4.1.6 Ein Kompromiss für die Praxis

Sie haben mehrere Methoden zur Verbreitung der Sitzungskennnummer kennen gelernt, die mehr oder minder effektiv alle auch in realen Szenarien funktionieren. Daneben gibt es möglicherweise noch andere Methoden, z.B. alle Ihre Seiten in einen einzigen Frame einzubetten und in den eingebetteten Seiten JavaScript zu verwerden, um auf die Sitzungskennnummer des Hauptframes zuzugreifen. Aber da Sie auf JavaScript und speziellen Layouteinstellungen basieren oder anderen Beschränken unterliegen, sind sie in der Regel nicht für professionelle Webanwendungen geeignet.

Für die Praxis empfehlen wir eine Kombination von Cookies und entweder dynamischen Pfaden oder DNS-Lösungen zu verwenden. Sofern machbar, erspart die Programmierung der Sitzungskennnummer in den Hostnamen dem Entwickler Arbeit.

In unserer Sitzung wurde die Sitzungskennnummer des Benutzers entweder in die URL oder den Hostnamen einprogrammiert. Bei PHP 4.0 ist die automatischen Änderung der URL auch eine Alternative, sofern auf der Website der Datenverkehr nicht allzu groß ist. Dies funktioniert mit allen Browsern, sowohl mit als auch ohne Cookies. Sorgen Sie dafür, dass Sie möglichst viele Benutzer erreichen. Wenn der Benutzer seine Cookies aktiviert hat, wird die ID auch als Cookies gespeichert. Beim nächsten Besuch des Benutzers wird er automatisch über den Cookie identifiziert und Sie können ihn mit einer per-

sönlichen Indexseite begrüßen, wie Amazon.com es tut. Wenn der Browser keine Unterstützung für Cookies bietet, muss der Benutzer sich mit einem Benutzernamen / Password anmelden. Danach kann er die Website wie jeder andere Besucher benutzen.

Zusammengefasst sollte die richtige Sitzungsverwaltung folgende Dinge tun:

- Sitzungsdaten auf dem Server speichern
- Eine willkürliche Sitzungskennnummer zur Identifizierung eines Benutzers verwenden
- Die Sitzungskennnummer (und nur die Sitzungskennnummer) mit Cookies, `GET/POST`, dem Skriptpfad oder DNS-Tricks auf der Clientseite speichern
- Im Idealfall automatisch andere Verfahren zur Verbreitung der Sitzungskennnummer anwenden, wenn der Benutzer seine Cookies deaktiviert hat

4.1.7 PHPs eigene Sitzungsbibliothek

Glücklicherweise besitzt PHP 4.0 eine integrierte Basis-Sitzungsverwaltung, wie Listing 4.1 zeigt. Diese ist recht einfach und direkt zu verwenden und reicht vielleicht für Ihre Bedürfnisse. Ihr fehlen aber die erweiterten Funktionen, die PHPLib bietet. Da PHPLib außerdem Module für die Verwaltung der Benutzerrechte und eine Datenbank-Abstraktionsebene hat, ist es immer noch eine sehr wichtige Bibliothek, die wir in Kapitel 6 »Datenbankzugriffe in PHP« behandeln.

```
// Start the session
session_start();

// Init the counter
if(!isset($counter))
{
     $counter = 0;
}

// Output session ID and counter
printf("Our session ID is: %s<br>", session_id());
print("The counter value is: $counter");

// Increment the counter
$counter++;

// Register our session variable
session_register("counter");
```

Listing 4.1: Einfaches Beispiel für die Verwendung von PHPs integrierten Sitzungen

Dieses Beispiel zeigt die Sitzungskennnummer und einen Zähler, der jedes Mal erhöht wird, wenn Sie auf die Seite zugreifen. Es unterscheidet sich natürlich von einem normalen Seitenzähler. Die Sitzung (und damit der Zähler) ist an einen speziellen Benutzer gebunden. Bei der Standardkonfiguration von PHP hat der Sitzungscookie eine Lebenszeit von 0; dies bedeutet, wenn Sie den Browser schließen und wieder öffnen, beginnt der Zähler wieder bei Null, da der Cookie gelöscht wurde.

Betrachten wir die Sitzungsfunktionen von PHP 4.0 näher. PHPs Sitzungsverwaltungsbibliothek bietet die bereits beschriebenen Optionen:

- Sie speichert Sitzungsdaten auf dem Server. Da die Bibliothek verschiedene Speichermodule verwendet, können Sie die Daten in reinen Textdateien, dem Shared Memory oder in Datenbanken ablegen. Dies spiegelt genau das wieder, was wir über Speichermedien erläutert haben. Wo die Daten aufbewahrt werden, ist nicht relevant (solange die Leistung des Mediums ausreicht).
- Sie verwendet eine willkürliche Sitzungskennnummer, um einen Benutzer zu identifizieren.
- Sie speichert die Sitzungskennnummer (und nur die Sitzungskennnummer) mittels Cookies, `GET`/`POST` oder dem Skriptpfad auf der Client-Seite. (Die PHP-Bibliothek bietet alle diese Methoden an; wie sie zu verwenden sind, zeigen wir später.)
- Wenn der Benutzer Cookies deaktiviert hat, kann das Programm andere Mittel für die Verbreitung der Sitzungskennnummer einsetzen.

Ablauf einer Sitzung

In PHP 4.0 beginnt eine Sitzung mit dem Aufruf von `session_start()` oder, indem Sie eine Sitzungsvariable mit `session_register()` registrieren. Beim Laden der Bibliothek prüft PHP, ob eine gültige Sitzungskennnummer vorhanden ist. Dazu führt es folgende Aktionen durch:

1. Wenn `track_vars` auf `false` gesetzt ist, überprüft die Bibliothek den globalen Namensraum auf eine Sitzungskennnummer. Wird eine gefunden, schickt die Bibliothek keinen Cookie mit der Sitzungskennnummer mehr, sondern definiert die Konstante `SID`.

2. Wenn `track_vars` aktiviert ist, aber keine Sitzungskennnummer im globalen Namensraum gefunden wurde, wird das Array `$HTTP_COOKIE_VARS` auf eine Sitzungskennnummer hin überprüft. Wird eine gefunden, dann wird weder ein Cookie gesendet, noch die Konstante `SID` definiert.

3. Wenn immer noch keine Sitzungskennnummer gefunden wurde, werden die Arrays `$HTTP_GET_VARS` und `$HTTP_POST_VARS` auf eine Sitzungskennnummer hin überprüft. Wird eine gefunden, wird die Konstante `SID` definiert.

4. Wenn immer noch keine Sitzungskennnummer gefunden wurde, wird der Pfad (`$REQUEST_URI`) in Hinblick auf eine Zeichenfolge der Form `<session-name>=<session-id>` analysiert. Wird eine solche gefunden, dann wird die Konstante `SID` definiert.
5. Wenn in der Client-Anfrage ein externer HTTP-Referer (von einem nicht lokalen Standort) angegeben und `extern_referer_check` (beachten Sie das einzelne »r«) in der PHP-Konfiguration aktiviert ist, wird die Sitzungskennnummer abgelehnt und als ungültig markiert. Mit dieser Maßnahme wird eine zusätzliche Sicherheit erreicht, da es verhindert, dass Benutzer von anderen PHP-Websites eine Sitzung übernehmen (was jedoch aufgrund des Algorithmus, der für die Generierung der Sitzungskennnummer verwendet wird, recht unwahrscheinlich ist).

Im Allgemeinen wird die Konstante `SID` immer dann definiert, wenn die Sitzungsbibliothek nicht sicher weiß, ob der Client Cookies unterstützt, in anderen Worten: solange keine Sitzungskennnummer im Array `$HTTP_COOKIE_VARS` gefunden wird.

Wenn bei all diesen Überprüfungen keine Sitzungskennnummer gefunden oder diese abgelehnt wurde, sollten Sie eine neue Sitzung starten, in der eine neue Sitzungskennnummer vergeben wird.

Wenn eine gültige Sitzungskennnummer vorhanden ist, werden die eingefrorenen Variablen der Sitzung aktiviert und in den globalen Namensraum wieder eingeführt. Sitzungsvariablen lassen sich genauso leicht handhaben wie `GET`/`POST` Variablen: Wenn Sie eine Variable namens `foo` registrieren, wird `$foo` nach dem Aufruf von `session_start()` automatisch als globale Variable zur Verfügung gestellt. Außerdem wird sie dem globalen Array `$HTTP_SESSION_VARS` hinzugefügt, sobald `track_vars` aktiviert wird. Nachdem die Funktion `serialize()` in PHP 4.0 verbessert wurde, können Sie auch Objekte (Klassen) als Sitzungsvariablen behandeln.

Aktivieren von track_vars und register_globals

Um die gesamte Funktionalität der Sitzungsverwaltungsbibliothek nutzen zu können, müssen Sie in Ihrer PHP-Konfiguration `track_vars` und `register_globals` aktiviert haben.

Alle Variablen, die Sie über mehrere Seitenanfragen hinweg erhalten möchten, müssen mit der Funktion `session_register()` in der Sitzungsbibliothek registriert werden. Beachten Sie, dass im Argument der Funktion nicht die Variable selbst, sondern der Namen der Variablen anzugeben ist. Die Variable `$foo` müssten Sie folgendermaßen eingeben:

```
session_register("foo");
```

Der Code

```
session_register($foo);
```

würde nur dann etwas Sinnvolles ausgeben, wenn $foo der Name einer anderen Variablen wäre:

```
$bar = "This is a string";
$foo = "bar";
session_register($foo);
```

Mit session_unregister() können sie Variablen aus der Sitzungsbibliothek entfernen.

Genau wie im wirklichen Leben lässt es sich nicht immer vorhersagen, wann eine Sitzung endet – es sei denn, durch ein gewaltsames Ende, das durch session_destroy() erzwungen wird. Wenn die Sitzung an Überalterung eingehen soll, müssen verschiedene Konfigurationen in Augenschein genommen werden. Wenn wir die Sitzungskennnummer über Cookies verbreiten, ist die Standardlebensdauer eines Cookies 0, d.h. er wird gelöscht, sobald der Benutzer den Browser schließt. Sie können die Lebensdauer mithilfe des Konfigurationswertes lifetime verändern. Da der Server nicht weiß, ob der Cookie auf der Client-Seite noch existiert, hat PHP eine weitere Lebenszeitvariable, die festlegt, wie lange die Daten nach dem letzten Zugriff auf diese Sitzung zerstört werden sollen: gc_maxlifetime. Wollte man jedoch so ein Cleanup alter Sitzungen (auch Speicherbereinigung genannt) bei jeder Seitenanfrage durchführen, würde dies zu einem erheblichen Overhead führen. Deshalb haben Sie die Möglichkeit anzugeben, mit welcher Wahrscheinlichkeit die Speicherbereinigungsroutine aufgerufen werden soll. Wenn gc_probability den Wert 100 hat, wird das Cleanup bei jeder Anfrage durchgeführt (d.h. mit einer Wahrscheinlichkeit von 100%). Wenn dieser, wie standardmäßig eingestellt, 1 ist, werden alte Sitzungen mit einer Wahrscheinlichkeit von 1% pro Anfrage gelöscht.

Wenn Sie keine Cookies verwenden, sondern statt dessen die Sitzungskennnummer per GET oder POST übergeben, müssen Sie auf die Speicherbereinigungsroutine besonders achten. Möglicherweise setzen Benutzer Lesezeichen für die URLs, welche die Sitzungskennnummer enthalten. Stellen Sie sicher, dass Sitzungen häufig bereinigt werden. Wenn die Sitzungsdaten noch existieren, wenn der Benutzer die Seite mit der Sitzungskennnummer zu einem späteren Zeitpunkt öffnet, nimmt er die vorherige Sitzung einfach wieder auf, anstatt eine neue Sitzung zu starten. Dies ist möglicherweise nicht in Ihrem Interesse. Setzen Sie gc_probability jedoch auch nicht auf einen zu hohen Wert, insbesondere wenn Sie Sitzungen in Dateien speichern. In diesem Falle werden bei der Durchführung der Speicherbereinigung mit stat() alle Sitzungsdateien genannt, und die letzte Änderungszeit dieser Sitzungen überprüft. Dies ist eine sehr teure Operation, die nicht allzu häufig gestartet werden sollte. In der Regel ist ein Wert zwischen 5 und 10 für gc_probability angemessen, insbesondere dann, wenn Sie Sitzungen löschen, sobald Sie eine Transaktion beendet haben (z.B. wenn sich ein Benutzer in Ihrem Shop abmeldet).

Speichermodule

Zum Lesen und Speichern von Sitzungsdaten verwendet PHP Speichermodule. Damit wird der Backend der Bibliothek abstrahiert. Derzeit gibt es drei Speichermodule: *files*, *mm* und *user*. Standardmäßig verwendet PHP das Modul *files*, um Sitzungsdaten auf der Festplatte zu speichern. Es erstellt eine Textdatei, die nach der Sitzungskennnummer benannt wird und legt sie in */tmp* ab. Für das vorhergehende Beispiel sähe der Inhalt der Datei folgendermaßen aus (hierbei handelt es sich um eine serielle Darstellung der Variablen):

```
counter|i:4;
```

Wahrscheinlich müssen Sie auf diese Datei nie direkt zugreifen.

Wenn Sie eine höhere Leistung benötigen, ist das Modul `mm` eine gute Alternative. Es speichert die Daten im Shared Memory und ist daher nicht durch das Ein-/Ausgabewerk der Hardware begrenzt. Das letzte Modul *user* wird intern verwendet, um Rückruffunktionen auf Benutzerebene durchführen zu können, die Sie mit `session_set_save_handler()` definieren.

Das wahre Potenzial liegt in der Fähigkeit, Benutzerrückrufe als Speichermodule zu spezifizieren. Da Sie Ihre Funktionen so schreiben können, dass sie Sitzungen verwalten und sich trotzdem auf die standardisierte PHP-API verlassen können, können Sie Sitzungen speichern wo und wie Sie wollen: in einer Datenbank wie MySQL, in XML-Dateien oder auf einem entfernten FTP-Server (okay, letzterer macht keinen Sinn, aber Sie verstehen, was wir meinen).

Die Funktion `session_set_save_handler()` hat sechs Argumente in Form von Zeichenfolgen. Diese fungieren als Ihre Rückruffunktionen. Die Funktion hat demnach folgende Syntax:

```
void session_set_save_handler(string open, string close, string read,
➥string write, string destroy, string gc)
```

Serialisierung von Daten

Serialisierung bezeichnet die Umwandlung von Variablen in eine Byte-Code-Darstellung. Diese kann überall als normale Zeichenkette gespeichert werden. Ohne diese Funktion wäre es beispielsweise nicht möglich, PHP-Arrays in einer Datenbank zu speichern. Die Serialisierung von Daten ist sehr nützlich, um Daten über mehrere Anfragen hinweg zu erhalten, eine wichtige Facette einer Sitzungsbibliothek. Sie können sowohl `serialize()` als auch `deserialize()` benutzen. Beachten Sie jedoch, dass diese Funktionen in PHP 3.0 nicht richtig für Objekte (Klassen) funktionieren. Klassenfunktionen werden verworfen.

Ausschluss von Argumenten
Um ein Argument wegzulassen, übergeben Sie eine leere Zeichenfolge ("") an `session_set_save_handler()`.

Die Funktionen sind wie folgt definiert:

- `bool open`(String `save_path`, String `sess_name`)
  Diese Funktion wird bei der Initialisierung einer Sitzung ausgeführt. Sie sollten sie verwenden, um Ihre Funktionen vorzubereiten, Variablen zu initialisieren o.ä. Die Funktion hat zwei Argumente in Form von Zeichenfolgen. Das erste gibt den Pfad an, in dem Sitzungen gespeichert werden sollten. Diese Variable kann in *php.ini* oder durch die Funktion `session_save_path()` angegeben werden. Sie können diese Variable als Platzhalter verwenden und sie für eine modulspezifische Konfiguration einsetzen. Das zweite Argument gibt den Namen der Sitzung an, der standardmäßig `PHPSESSID` lautet. Die Funktion `open()` gibt bei erfolgreicher Initialisierung `true` zurück, bei nicht erfolgter Initialisierung `false`.
- `bool close()`
  Diese Funktion wird beim Beenden einer Sitzung ausgeführt. Verwenden Sie sie, um Speicher freizugeben oder Ihre Variablen zu löschen. Die Funktion hat keine Argumente. Sie gibt `true` aus, wenn die Sitzung erfolgreich geschlossen wurde, ansonsten `false`.
- `mixed read`(String `sess_id`)
  Diese wichtige Funktion wird aufgerufen, wenn eine Sitzung gestartet wird. Sie muss die Daten der Sitzung auslesen, die mit `sess_id` gekennzeichnet ist, und diese als serielle Zeichenfolge zurückgeben. Wenn es keine Sitzung mit dieser Kennnummer gibt, sollte eine leere Zeichenkette "" zurückgegeben werden. Bei einem Fehler lautet die Ausgabe `false`.
- `bool write`(String `sess_id`, String `value`)
  Wenn die Sitzung gespeichert werden muss, wird diese Funktion aufgerufen. Das erste Argument ist eine Zeichenfolge, welche die Sitzungskennnummer angibt. Das zweite Argument ist die serielle Darstellung der Sitzungsvariablen. Bei erfolgreicher Speicherung gibt der Wert `true` aus, bei einem Fehler der Wert `false`.
- `bool destroy`(String `sess_id`)
  Wenn der Entwickler `session_destroy()` aufruft, wird diese Funktion ausgeführt. Sie löscht alle Daten, die der Sitzung mit der Kennnummer `sess_id` zugeordnet sind, und gibt bei erfolgreicher Löschung `true` zurück, ansonsten `false`.
- `bool gc`(Int `max_lifetime`)
  Diese Funktion wird beim Einrichten einer Sitzung mit der Wahrscheinlichkeit aufgerufen, die in `gc_probability` angegeben ist. Sie wird zur Speicherbereinigung verwendet. Dabei werden alle Sitzungen entfernt,

die seit mehr als `gc_maxlifetime` Sekunden nicht mehr aktualisiert wurden. Hier wird bei einer erfolgreichen Speicherbereinigung `true` zurückgegeben, bei einem Fehler `false`.

Das Beispiel in Listing 4.2 aktiviert die Benutzerrückruffunktionen und definiert ein Speichermodul, das die Sitzungsdaten in einer MySQL-Datenbank ablegen soll. (Das komplette Beispiel einschließlich dem erforderlichen MySQL-Tabellenschema befindet sich auf der CD-ROM, die mit dem Buch mitgeliefert wird.) Da `session_set_save_handler()` derzeit keine Klassenfunktionen, sondern nur einfache Funktionen akzeptiert, haben wir anstatt einer Klasse die gute alte strukturelle Programmierung verwendet. Vererbung und mehrere Instanzen würden für diese Art von Code sowieso keinen Sinn ergeben, daher ist es hier kein großer Verlust.

```
$sess_mysql = array();
$sess_mysql["open_connection"]  = true;
➥// Establish a MySQL connection on session startup?
$sess_mysql["hostname"]          = "localhost";  // MySQL hostname
$sess_mysql["user"]              = "root";       // MySQL username
$sess_mysql["password"]          = "";           // MySQL password
$sess_mysql["db"]                = "book";       // Database where to
                                                         ➥ store the
sessions
$sess_mysql["table"]             = "sessions";   // Table where to store
                                                     ➥ the sessions

function sess_mysql_open($save_path, $sess_name)
{
    global $sess_mysql;

    // Establish a MySQL connection, if $sess_mysql["open_connection"]
       ➥is true
    if ($sess_mysql["open_connection"])
    {
        $link = mysql_pconnect($sess_mysql["hostname"], $sess_
        ➥mysql["user"],  $sess_mysql["password"]) or die(mysql_
error());
    }

    return(true);
}

function sess_mysql_read($sess_id)
{
    global $sess_mysql;
    // Select the data belonging to session $sess_id from MySQL session
```

```
            ➥table

    $result = mysql_db_query($sess_mysql["db"], "SELECT data FROM
    ➥".$sess_mysql["table"]." WHERE id = '$sess_id'") or die(mysql_
error());

    // Return an empty string if no data was found for this session
    if(mysql_num_rows($result) == 0)
    {
        return("");
    }
    // Session data was found, so fetch and return it
    $row = mysql_fetch_array($result);
    mysql_free_result($result);

    return($row["data"]);
}

function sess_mysql_write($sess_id, $val)
{
    global $sess_mysql;

    // Write the serialized session data ($val) to the MySQL session
       ➥table
    $result = mysql_db_query($sess_mysql["db"], "REPLACE INTO
    ➥".$sess_mysql["table"]."VALUES ('$sess_id','$val', null)")
    ➥or die(mysql_error());

    return(true);
}

function sess_mysql_destroy($sess_id)
{
    global $sess_mysql;

    // Delete from the MySQL table all data for the session $sess_id
    $result = mysql_db_query($sess_mysql["db"], "DELETE FROM
    ➥".$sess_mysql["table"]." WHERE id = '$sess_id'") or die(mysql_
    ➥error());

    return(true);
}

function sess_mysql_gc($max_lifetime)
{
    global $sess_mysql;
```

```
    // Old values are values with a Unix less than now - $max_lifetime
     $old = time() - $max_lifetime;
    // Delete old values from the MySQL session table
     $result = mysql_db_query($sess_mysql["db"], "DELETE FROM
     ➥".$sess_mysql["table"]." WHERE UNIX_TIMESTAMP(t_stamp) < $old") or
     ➥die(mysql_error());

     return(true);
}

/*
 *  Basic Example: Registering above functions with session_set_save_
    ➥handler()
 *
$foo = 10;
session_set_save_handler("sess_mysql_open", "", "sess_mysql_read",
➥"sess_mysql_write", "sess_mysql_destroy", "sess_mysql_gc");
session_start();
session_register("foo");
echo "foo: $foo";
$foo++;
 *
 */
```

Listing 4.2: Ein MySQL-Speichermodul für die PHP 4.0-Sitzungsbibliothek

Seiten-Caching

Mit der Sitzungsbibliothek können Sie auch kontrollieren, wie Seiten gecacht werden. Hierzu steht Ihnen die HTTP-Anweisung HTTP `Cache-Control` zur Verfügung. In der PHP-Konfiguration kann die Anweisung `cache_limiter` auf `nocache`, `private` oder `public` gesetzt werden. Wie wir in Kapitel 6 zeigen werden, ähnelt dies dem Verhalten von PHPLib (beachten Sie jedoch, dass PHPLib `no` anstelle von `nocache` verwendet).

Das Seiten-Caching ist standardmäßig auf `nocache` gesetzt. Damit wir Caching vollständig verhindert, was – wie Sie vielleicht wissen – dem Standardverhalten aller PHP-Seiten entspricht. Bei dynamisch erstellten Seiten sollten Sie diese Einstellung bevorzugt verwenden, da diese Seiten häufig von Anfrage zu Anfrage differieren. Bei einigen Teilen Ihres Programms, die sich nicht häufig ändern, möchten Sie möglicherweise von diesem Ansatz abweichen. Ihre Serverhardware wird es Ihnen danken. Der Kopf der Ausgabe sieht folgendermaßen aus:

```
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-cache
Pragma: no-cache
```

Wenn Sie `cache_limiter` auf `private` setzen, können die Seiten zwar von Browsern, aber nicht von Proxys oder anderen Gateway-Programmen gecacht werden. Betrachten Sie den Unterschied zur Anweisung `proxy-revalidate`. In letzterem Fall merkt sich der Proxy den Inhalt der Seite und stellt diese wieder her, anstatt sie vollständig neu aufzurufen. Der generierte HTTP-Kopf sieht etwas so aus:

```
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: private, max-age=10800
Last-Modified: Thu, 03 Feb 2000 15:56:11 GMT
```

Der letzte Wert `public` ermöglicht sowohl dem Client als auch Proxys vollständiges Caching. Verwenden Sie die Cache-Option `public` jedoch mit Vorsicht. Seiten, die mit dieser Einstellung generiert werden, könnten auch anderen Benutzern zugänglich sein, die Zugriff auf Proxys haben.

Bei der öffentlichen Variante wird auch der PHP-Konfigurationsbefehl `cache_expire` betrachtet. Dieser gibt an, nach wie vielen Sekunden der Cache abläuft. Die erzeugten Köpfe sollten in etwa so aussehen:

```
Expires: Thu, 03 Feb 2000 18:56:11 GMT
Cache-Control: public, max-age=10800
Last-Modified: Thu, 03 Feb 2000 15:56:11 GMT
```

PHP 3.0-Sitzungen

Da sich die Anwenderprogrammierschnittstelle für die PHP 4.0-Sitzungen so einfach und intuitiv verwenden lässt, wollten wir dies auch für PHP 3.0 realisieren. Eine konsistente Sitzungsschnittstelle für beide Versionen wäre großartig, dachten wir. Für einige kleinere Projekte möchten man einfach nicht PHPLib verwenden. Deshalb sahen wir uns `sessions.c` näher an und portierten es nach PHP 3.0. Einige Interna, wie etwa der Algorithmus für die Generierung der Sitzungskennnummer, behielten wir nicht bei, aber wir versuchten, die API zu 100% kompatibel zu PHP 4.0 zu machen. Es gibt jedoch einige offensichtliche Beschränkungen. So ist eine automatische Änderung der URL nicht möglich. Wenn Sie die dokumentierten Unterschiede im Hinterkopf behalten, sollten Sie eigentlich mit der Bibliothek als ad hoc-Ersatz der PHP 4.0-Sitzungsfunktionen zurechtkommen. Den vollständigen Quellcode finden Sie auf der CD-ROM.

Unser Experiment hilft Ihnen eventuell auch, die Funktionsweise der interne Sitzungsbibliothek von PHP besser zu verstehen. Die Funktion `session_start()` spiegelt beispielsweise die ursprüngliche C-Funktion sehr genau wieder.

4.2 Sicherheitsaspekte

Beispiel 1: Anfang 1999 führten wir einen Website-Audit für eine führende Online-Jobdatenbank durch. Während der Analyse entdeckten wir einen Sicherheitsaspekt, der uns sprachlos machte. Die Website war mit einem Onlineshop verknüpft, in welchem sich Besucher Bücher zum Thema Jobhunting bestellen konnten. Der Shop wurde von einem unabhängigen Auftragnehmer entwickelt, der bereits mehrere andere Onlinegeschäfte entwickelt hatte, die alle auf seinen Pearl-Skripten basierten. Hiervon hat er auf seiner Website ein Demo zur Verfügung gestellt, einschließlich wichtiger Teile des Quellcodes. Jedes Mal, wenn ein Besucher einen Artikel bestellte, wurde eine reine Textdatei erstellt, die alle Bestelldaten enthielt, selbst Kreditkartendaten – und das alles unverschlüsselt. Nicht genug, dass die Dateien auf einem für jedermann sichtbaren Verzeichnis auf dem Webserver abgelegt waren – »um es den Website-Managern einfacher zu machen«, wie er uns erzählte. Das Verzeichnis wurde nur dadurch »geschützt«, dass es in der Apache-Konfiguration als »nicht browserfähig« deklariert wurde. Da die Dateinamen nach einer Standardnamenskonvention der Form »yyyy-dd-mm-hh-mm-ss.txt benannt wurden, wäre es für einen Hacker kein Problem, ein Skript zu schreiben, um Dateien zu suchen.

Beispiel 2: Network Solutions, die für lange Zeit einziger Domänenbesitzer waren, beschlossen im September 1999, ihren bevorzugten Kunden einen freien Web-Mailaccount zur Verfügung zu stellen. Sie schufen für jedes Konto einen Benutzernamen sowie ein Passwort und sandten es den entsprechenden Kunden zu. Der Benutzername bestand aus dem Nachnamen des Kunden (»doe«), und das Passwort setzte sich aus dem Benutzernamen und der Endung »nsi« zusammen (»doensi«). Über 24 Stunden konnten sich Benutzer im System anmelden, indem sie die Passwörter anderer Kunden eingaben. Sie konnten das Passwort ändern, Mails lesen, die zu diesem Account geschickt wurden, und selbst Mails im Namen dieser Kunden verschicken.

Beide Sicherheitsprobleme entstanden nicht durch Fehler, die durch die Unzulänglichkeit der Programmiersprache hervorgerufen wurden, sondern durch eine schlechte Programmierung. PHP ist an sich sehr sicher. Wir haben niemals von Löchern wie ASP's `::$DATA` -Fehlern gehört. Bei diesem im Juni 1998 entdeckten Phänomen konnte jeder Benutzer den Quellcode von ASP-Skripten im Web ansehen, indem er einfach die Zeichenkette `::$DATA` an die URL der Datei anhängte, (z.B. `www.server.com/script.asp::$DATA`) – ein Alptraum für jeden Softwareentwickler. Ein früheres, nur Windows betreffendes Sicherheitsproblem entstand, als Benutzer auf die Quelle zugreifen konnten, indem Sie eine URL wie `www.server.com/script.asp.` angaben (beachten Sie den Punkt am Ende). Nach Auftauchen des `::$DATA`-Fehlers konnte man auf einigen Versionen des Personal-Webserver für Windows die Form `www.server.com/....../` usw. benutzen, um nicht nur auf den Dokumentenstammbaum des Webservers, sondern auf die gesamte Festplatte zugreifen.

Bei Verwendung einer Skriptsprache muss bereits im aller ersten Schritt, d.h. der Installation, auf die Sicherheit geachtet werden. Wie Sie wissen kann PHP als Servermodul (Apache-Modul), ISAPI / NSAPI-Plugin oder eigenständiges CGI-Programm installiert werden.

Wenn Sie es als Servermodul installieren, ist es Teil des zugrunde liegenden Webservers und erbt seine Sicherheit. Für diese Art Einstellungen sind keine PHP-spezifischen Attacken bekannt. Natürlich brauchen Sie noch einen sicheren Server, aber dies ist ein zu weites Feld, als dass wir es hier behandeln könnten.

CGI-Programme bieten dagegen eine hervorragende Angriffsfläche für eine Vielzahl möglicher Attacken, sowohl Einbruchs- als auch Dienstverweigerungsattacken. Traditionell entstehen die größten Probleme, wenn Sie einen Skriptinterpreter in das Verzeichnis *cgiwin* eines Webservers einfügen. Bei einigen Skriptinterpretern können Benutzer jeden Befehl über den Server schicken. PHP versucht einige dieser Attacken zu verhindern. Wenn es vom Web aus gestartet wird, verwirft PHP Befehlszeilenparameter, die von der CGI-Schnittstelle kommen – Anfragen wie etwa `http://server.com/cgi-bin/php?etc/passwd` schlagen fehl.

Unglücklicherweise kann eine andere Attacke, die direkt mit Mängeln in der CGI-Spezifikation zusammenhängt, immer noch durchkommen. Sie können auf jede Datei unterhalb des Webserver-Dokumentenstamms zugreifen, selbst wenn das Verzeichnis durch `HTTP-AUTH` (eine *.htaccess*-Datei) geschützt ist. Dazu müssen Sie lediglich über den PHP-Interpreter: `www.server.com/cgi-bin/php/secret-directory/file.html` aufrufen. HTML ermöglicht Ihnen sogar `file.html` anzusehen, wenn `secret-directory` durch eine *.htaccess*-Datei geschützt ist. Auf dem Apache-Server können Sie dieses Problem vermeiden, indem Sie bei der Kompilierung von PHP `--enable-force-cgi-redirect` aktivieren Ausführliche Information zu diesem Thema finden Sie im PHP-Handbuch im Installationsabschnitt.

Die Wahl einer sicheren Installation ist allerdings nur der erste Schritt. Ein guter Programmierer behält die Sicherheit während des gesamten Entwicklungsprozesses im Auge. Es gibt so viele potenzielle Risiken, dass wir nicht alle in diesem Kapitel behandeln können. Wir können lediglich versuchen, Ihnen ein paar allgemeingültige Tipps zu geben.

4.2.1 Vertrauen Sie dem Web nicht

Alle Daten, die vom Web kommen, sind unsicher und sollten von Ihrem Programm überprüft werden. Es gibt beispielsweise keine Garantie, dass Ihr Skript über die passende Formular-Schnittstelle aufgerufen wurde. Benutzer könnten das HTML-Formular einfach umgehen und das Skript direkt aufrufen, indem sie möglicherweise über ihr eigenes `GET` oder `POST` Parameter spezifizieren.

Die Überprüfung von Formulardaten ist eine der ermüdendsten Aufgaben, die ein Webanwendungs-Entwickler bei seiner täglichen Arbeit erledigen muss. Die grundlegenden Überprüfungen können mit Routinen der Bibliothek automatisiert werden, die wir in Kapitel 5 »Grundlegende Webanwendungsstrategien« vorstellen. Für komplexere Überprüfungen haben wir leider noch keinen allgemeinen Ansatz gefunden. Daher führen wir sie manuell durch. Die entsprechenden Skripte folgen der typischen Logik, die wir in Kapitel 5 erläutern und die wir als PHP-Standardformular bezeichnen.

Ausführen von Systembefehlen

Achten Sie besonders auf die Sicherheit, wenn Sie mit Dateien arbeiten oder Systembefehle ausführen. Stellen Sie sich ein typisches Skript für einen Quellcode-Viewer vor, in dessen Argument der Dateiname angegeben wird und das die Datei farbig anzeigt.

```
show_source($Datei);
```

Sie möchten, dass das Skript mit `script.php3?file=script.php3` aufgerufen wird? Aber was passiert, wenn jemand es mit `script.php3?file=/etc/passwd` aufruft? Richtig – Sie haben ein Problem, weil Sie darauf vertrauten, dass sich die Variablen, die vom Web kommen, in einem bestimmten Bereich befinden (z.B. dem aktuellen Verzeichnis). Diese Annahmen müssen Sie dem Server unbedingt mitteilen, z.B. indem Sie folgenden Befehl verwenden.

```
show_source(basename($file));
```

Sehen wir uns ein anderes Beispiel an, einen Verzeichnis-Viewer, den wir auf dem Web gefunden haben. Listing 4.3 zeigt eine leicht geänderte und gekürzte Version. Der Autor Marcus Xenakis gab uns freundlicherweise die Genehmigung, ihn hier abzudrucken.

```
print("<pre>");
exec("ls -la $dir", $lines, $rc);
$count = count($lines) - 1;
for ($i = 1; $i <= $count; $i++)
{
    $type = substr($lines[$i], 0, 1);
    $name = strrchr($lines[$i], " ");
    $name = substr($name, 1);
    $dire = substr($lines[$i], 0, strpos($lines[$i], $name));
    printf('<font color="%s">', ($type == d) ? "blue" : "black");
    print("$dire</font>");
    if ($type == "d")
    {
        if ($name == "." or $name == "..")
        {
            print("$name<br>");
```

```
        }
        else
        {
            printf("<a href=\"".basename($PHP_SELF)."?dir=%s\">$name</
            ➥a><br>",  empty($dir) ? $name : "$dir/$name");
        }
    }
    else
    {
        printf("<a href=\"%s\">$name</a><br>", _
        ➥empty($dir) ? $name : "$dir/$name");
    }

}

print("</pre>");
```

Listing 4.3: Verzeichnisanzeige mit Sicherheitsrisiken

Er bestätigte zwar unsere Annahme, dass das Skript in eine zuverlässige Umgebung eingebettet werden sollte. Es gibt jedoch einige Techniken, die es zu einem gefährlichen Sicherheitsleck machen würden, wenn ein unerfahrener Benutzer das Skript in einem öffentlich zugänglichen Verzeichnis platzierte. Rufen Sie es doch einmal mit `Directory_Viewer.php3?dir=/etc` auf. Nett, nicht wahr? Sie können jedes Verzeichnis auf dem System durchblättern, für das PHP zugelassen ist. Aber das ist noch nicht alles. Sie können mithilfe des kleinen Skripts jeden Befehl ausführen und damit bis auf das Wurzelverzeichnis des Servers zugreifen, von dem es stammt.

Der wichtigste Abschnitt ist folgende Zeile:

```
exec("ls -la $dir", $lines, $rc);
```

Die Variable `$dir` wird, sofern sie dem Benutzer zur Verfügung steht, direkt an `exec()` weitergeleitet. Wie Sie vielleicht wissen, können Sie Shell-Befehle durch Semikolon miteinander verknüpfen. Was glauben Sie also wird passieren, wenn `$dir` gleichzusetzen ist mit *»/etc; cat/etc/passwd«?* Wenn Sie diese Zeichenfolge als Argument übergeben wollten, müssten Sie sie mit URL kodieren, so dass der Skript folgendermaßen aufgerufen würde:

```
Directory_Viewer.php3??dir=/etc%3B+cat+%2Fetc%2Fpasswd
```

Und damit würde tatsächlich der Inhalt von */etc/passwd* angezeigt. Anstelle des Befehls `cat` könnten Sie auch jeden anderen Befehl ausführen, z.B. `fetch`, und damit ein trojanisches Pferd von Ihrem eigenen Server holen und installieren. Abhilfe für dieses spezielle Problem schaffen Sie, wenn Sie die Variable `$dir` mit `EscapeShellCmd()` übergeben. Damit werden alle kritischen Zeichen

maskiert, die man verwenden könnte, um die Shell auszutricksen und verknüpfte Befehle auszuführen. Außerdem könnten Sie den Befehl soweit einschränken, dass er nur Unterverzeichnisse auflistet.

```
$secure_dir = str_replace(".", "", $dir);
$secure_dir = $DOCUMENT_ROOT.dirname($PHP_SELF)."/$secure_dir";
$secure_dir = EscapeShellCmd($secure_dir);
```

Das Prinzip bleibt das gleiche. Vertrauen Sie niemals Variablen, die von Benutzern kommen. Dies gilt natürlich für alle Skriptsprachen, nicht nur für PHP. Dasselbe Loch gibt es auch in ASP über das Objekt `FileSystem` oder in Pearl bei der Ausführung von benutzerdefinierten Befehlen.

Belastete Variablen

Wir können es nur noch einmal betonen: Alle Daten, die aus dem Benutzerraum kommen, müssen als belastet, nicht vertrauenswürdig, kontaminiert und potenziell übel betrachtet werden. In diesem Fall befindet sich das Internet außerhalb des Programmbereichs. In Bezug auf Vertrauen wird dies als Vertrauensgrenze bezeichnet. Der Anwendungsraum ist eine vertraute Umgebung, das Internet nicht. Die Übergabe der Daten von Ihrem Programm zum Client bedarf keiner besonderen Aufmerksamkeit, vorausgesetzt, dass das Programm seine Daten von zuverlässigen Systemen erhält. Das Datenbanksystem sollte das gleiche Vertrauen genießen wie das Programm selbst. Spezielle Vorkehrungen sind erforderlich, wenn Sie sicherstellen wollen, dass die Daten nur von einem speziellen Client empfangen werden oder dass der Client sicher sein kann, die Daten von einer speziellen Instanz (Ihrem Server) zu erhalten. Bei einem normalen HTTP-Transfer können Sie dies nicht gewährleisten. In diesem Fall sollten Sie SSL oder eine gleichwertige Verschlüsselungsschicht verwenden.

Bei der Weiterleitung von Daten von einer niedrigeren Sicherheitsebene zu einer höheren (wie beim Import von Benutzervariablen) ist größere Sorgfalt erforderlich. Sie können nicht davon ausgehen, dass die gelieferten Daten irgendwelchen Anforderungen entsprechen – nicht einmal, wenn Sie die Daten zum Client zuerst liefern. Sie könnten beispielsweise auf der Client-Seite Daten in einem HTML-Formular mit JavaScript überprüfen. Aber Sie können nicht davon ausgehen, dass die Daten auf dem Server das erwartete Format haben, da der Benutzer JavaScript ausgeschaltet haben könnte oder das Formular über Telnet abgeschickt hat. Ein anderer weitverbreiteter Fehler besteht darin, dass viele Daten zum Benutzer schicken und sicher davon ausgehen, dass sie nicht geändert werden. Zum Beispiel könnte eine Seite Kontodaten für einen Benutzer anzeigen, die mit einer Abfragezeichenfolge wie `script.php3?user_id=1` aufgerufen werden, nachdem sich der Benutzer angemeldet hat. Was hält den Benutzer davon ab, die Variable `user_id` in etwas anderes als 1 umzuwandeln und die Daten eines anderen zu bearbeiten?

Viele Webanwendungen überprüfen bereits den Inhalt, der von einem Benutzer für einen anderen Benutzer zur Verfügung gestellt wird. Sie werden kaum ein Nachrichtenbrett finden, in dem Sie `<script>`-Tags oder ähnlichen HTML-Code eingeben können. Entwickler machen häufig den Fehler, nur diese Art von Daten zu überprüfen und vernachlässigen dabei Daten, die nur vom Client selbst benutzt werden. Warum sollte ein Benutzer auch böswilligen Programmcode eingeben, den nur der Benutzer sieht?

Der Punkt ist, dass die »Infizierung« eines Programms mit einem Skript- oder einem HTML-Code eine ernsthafte Vertrauensverletzung darstellt. Der böswillige Code wird mit der Sicherheitsebene des Programms ausgeführt, denn er scheint ja vom Programm selbst zu kommen.

Häufig können Benutzer beim Einsehen von Inhalten, die ursprünglich nur für die Augen eines anderen Benutzers bestimmt waren, hereingelegt werden. Ein Beispiel: Nehmen wir an, Sie haben eine Suchmaschine namens `<script>alert("Hello World")</script>` konstruiert. Ein Benutzer gibt einen Schlüsselbegriff ein, und die Suchmaschine sucht im Web danach. Bei der Planung gingen Sie davon aus, dass nur der Benutzer, der die Suche tatsächlich gestartet hat, die Ergebnisse sehen wird. Deshalb kümmerten Sie sich nicht darum, in den Schlüsselbegriffen bestimmte Zeichen zu kodieren. Ein Benutzer könnte `<script>alert("Hello World")</script>` als Schlüsselbegriff eingeben und bekäme im Browser eine JavaScript-Meldung angezeigt (sofern JavaScript aktiviert ist). Solange die Benutzer die Suchbegriffe selbst eingeben, gibt es keine großen Probleme. Das Schlimmste, was passieren kann, ist, dass ihr eigener Browser aufgrund von Fehlern in JavaScript abstürzt.

Doch warten Sie. Warum sollten Benutzer nicht andere auf die Ergebnisse einer bestimmten Suche hinweisen, die sie für nützlich erachten. Unter `phpWizard.net` finden Sie beispielsweise ein Formular, das automatisch Amazon nach allen PHP-verwandten Büchern durchsucht.

Nun wird die Sache haarig. Ein Hacker kann auf seiner öffentlichen Website einen Link zu Suchergebnissen für den Begriff `<script>alert("Hello World")</script>` haben. Alle Benutzer, die diesen Link nutzen oder das Suchformular abschicken, erhalten die gemeine »Hello World«-Nachricht als Popup-Meldung auf ihrem Browser. Man könnte natürlich noch viel gefährlichere Dinge tun, als Nachrichten auf den Bildschirm zu bringen. Wenn wir das Beispiel ein wenig erweitern, können wir PHP als Suchmaschine für eine E-Commerce-Website verwenden, die eine eigene Sitzungsverwaltung verwendet und die Sitzungskennnummer in Cookies speichert. Wenn wir nun von einem intelligenten Hacker ausgehen, wird er anstelle der »Hello World«-Nachricht ein anderes JavaScript verwenden, um die Cookie-Daten zu lesen, die er dann an seine Website schickt. Dort wartet er auf ankommende Sitzungskennnummern, übernimmt die Sitzungen der anderen Benutzer und kauft ein paar nette Geschenke für die Leute unter `phpWizard.net`.

Vielleicht sind wir gut im Geschichten erfinden, aber dies könnte tatsächlich passiert sein. Die Produktsuchmaschine von Amazon hatte Tags nicht richtig kodiert – bis entsprechende Sicherheitsratschläge von CERT ausgegeben wurden. Diese finden Sie unter `www.cert.org/advisories/CA-2000-02.html`.

Selbst wenn wir uns all diese Ratschläge merken und alle vom Benutzer gelieferten Variablen überprüfen, können wir sehr leicht die falschen Überprüfungen vornehmen. Bei einigen Programmen sollen beispielsweise bestimmte HTML-Tags in Daten zugelassen sein. Eines dieser Tags ist das Tag `<p>`, das die Formatierung von Texten in Absätzen ermöglicht. Es kann ein Attribut namens `align` haben, das die Ausrichtung des Absatzes angibt. Um das öffnende Tag anzupassen, sollten Sie in einem ersten Versuch den regulären Ausdruck `<p[/>]>` verwenden. Viele Browser unterstützen aber auch für eine ganze Reihe von Tags das allgemeine Skriptverhalten. Benutzer können ein beliebiges JavaScript senden, das im Ereignis `onClick` oder `onMouseOver` des Tags `<p>` eingebettet ist, und den gemeinen Code erneut ausführen.

Zunächst möchten wir Ihnen klarmachen, dass all diese Bedrohungen zusammengenommen ein sehr hässliches Bild ergeben. Sie müssen wirklich sehr vorsichtig sein, wenn Sie alle Fallen vermeiden möchten. Dies ist auch der Hauptgrund weswegen wir empfehlen, einen Sicherheitsberater in das Entwicklungsteam aufzunehmen.

Hier einige allgemeine Hinweise und Richtlinien um diese Risiken zu minimieren:

- Verwenden Sie Sitzungen, anstatt Daten auf dem Client von einer Seite auf die nächste weiterzuleiten.
- Überprüfen Sie alle Daten aus dem Benutzerraum inklusive Kodierung bzw. Ersetzen des kleiner als- (<) und größer als-Zeichens (>) sowie des Ampersands (&). Achten Sie auch auf doppelte Anführungsstriche (»), einfache Anführungsstriche(l') und Leerräume, zumindest in Tag-Attributen und Attributwerten.
- Stellen Sie sicher, dass Ihr Programm in einer zuverlässigen Umgebung arbeitet.
- Achten Sie besonders auf die Reihenfolge der PHP-Variablen (siehe nächster Abschnitt).

Reihenfolge der PHP-Variablen

Wie Sie wissen, stellt PHP alle `GET`- und `POST`-Variablen automatisch im globalen Namensraum zur Verfügung. Wussten Sie, dass Sie diese Funktion in PHP 4.0 abschalten können?

Die automatische Einführung einer Variablen ist zwar eine Funktion, welche die Verwendung von PHP für unerfahrene Benutzer vereinfacht; sie kann

jedoch in größeren und komplexeren Programmen problematisch werden. Wenn Sie auf Variablen aus dem globalen Namensraum zugreifen, die vom Benutzer übergeben wurden, können Sie nicht sicher sein, woher sie wirklich kommen: GET, POST oder Cookies?

Wenn ihnen die Reihenfolge der Variablen nicht wichtig ist, akzeptieren Sie, dass jeder Benutzer Ihr Skript sowohl mit GET als auch mit POST aufrufen kann. Auch wenn es kein Sicherheitsproblem darstellt, ist es schlechter Stil. Sie sollten wählen können, wie die Daten zu Ihrem Programm übertragen werden. PHP bietet eine Methode, um auf Variablen eines bestimmten Namensraums zuzugreifen. Wenn track_vars in der HPP-Konfiguration aktiviert wurde, können Sie für jeden Namensraum auf ein assoziatives Array zugreifen. Die folgende Tabelle zeigt die verfügbaren Arrays:

Array-Name	Inhalt
$HTTP_GET_VARS	Variablen aus einer GET-Anfrage
$HTTP_POST_VARS	Variablen aus einer POST-Anfrage
$HTTP_COOKIE_VARS	Variablen aus Cookies
$HTTP_ENV_VARS	Umgebungsvariablen, z.B. $SHELL
$HTTP_SESSION_VARS	Sitzungsvariablen
$HTTP_SERVER_VARS	Servervariablen, in unserer Box $argc und $argv

Beachten Sie, dass PHP 3.0 nur die ersten drei Arrays kennt.

Von einigen cleveren Projektmanagern weiß man, dass sie den PHP-Konfigurationsbefehl register_globals (nur in PHP 4.0 verfügbar) auf false setzen, um ihre Programmierer zu zwingen, die $HTTP_*_VARS-Arrays zu verwenden.

Sie können auch die Reihenfolge beeinflussen, in welcher Variablen im globalen Namensraum hinzugefügt werden. Standardmäßig wird der Konfigurationsbefehl variables_order auf »EGPCB« gesetzt. Dies weist PHP an, die Variablen in folgender Reihenfolge zu integrieren:

1. Umgebungsvariablen
2. GET
3. POST
4. Cookies
5. Interne Variablen (Servervariablen)

Hierbei überschreiben neuere Werte die älteren. Bei der Übergabe einer PATH-Variablen in der GET-Anfrage eines Benutzers würde demnach die Umgebungsvariable überschreiben. Durch Verwendung von getenv() oder Ände-

rung des Befehls `variables_order` können Sie sicherstellen, dass Sie tatsächlich auf Umgebungsvariablen und nicht etwa auf vom Benutzer gelieferte Variablen zugreifen.

Sitzungsvariablen überschreiben stets Variablen, die von anderen Bereichen kommen. Da sie aus einer bereits gesicherten Vertrauenszone stammen, entfallen damit viele Sicherheitsprobleme.

4.2.2 Erfinden Sie die Kryptografie nicht neu

Kryptografie ist die Wissenschaft für die Verwendung von Formeln zur Verschlüsselung bzw. Entschlüsselung von Daten. Sie ermöglicht Ihnen, sensible Daten zu speichern oder über unsichere Kommunikationskanäle zu übertragen, und zwar so, dass sie von keinem anderem als dem beabsichtigten Empfänger gelesen werden können. Datenverschlüsselung ist eine Wissenschaft für sich. Versuchen Sie erst gar nicht, einen eigenen Verschlüsselungsalgorithmus zu erfinden. Verwenden Sie bewährte Algorithmen, wie etwa RC5 oder Blowfish.

Verschlüsselung mit Mcrypt-Funktionen

Wenn Sie PHP mit dem Modul `mcrypt` kompiliert haben, stehen Ihnen eine Vielzahl von mächtigen Verschlüsselungs- und Entschlüsselungsalgorithmen zur Verfügung. Der Abschnitt »Die Mcrypt-Funktionen« zeigt Ihnen, wie Sie dieses Modul verwenden und wie Sie herausfinden, welche Algorithmen auf Ihrem System unterstützt werden.

Es gibt zwei Arten von Verschlüsselung: Symmetrische Verschlüsselung und Verschlüsselung über öffentliche Schlüssel.

Symmetrische Verschlüsselung

Die symmetrische Verschlüsselung wird auch als Verschlüsselung per geheimen Schlüssel bezeichnet. Sie verwendet denselben Schlüssel für die Verschlüsselung und die Entschlüsselung der Daten. Der Datenverschlüsselungsstandard (Data Encryption Standard) DES ist ein gutes Beispiel für diese Methode. Hierbei handelt es sich um einen komplexen Algorithmus, der in den 70er Jahren von IBM entwickelt und 1976 vom U.S. Bureau of Standards genehmigt wurde. Es ist zwar relativ einfach, diesen 56-Bit-Algorithmus zu knacken (die DES Challenge III, ein Knackversuch, der von der RSA Data Security gesponsert wurde, benötigte nur 22 Stunden bis die verschlüsselte Nachricht dechiffriert war), aber er kann trotzdem für die Verschlüsselung von nicht-kritischen Daten verwendet werden. Einige Daten müssen einfach nur vor den normalen Systembenutzern »versteckt« werden und brauchen nicht auf kryptografisch sichere Weise verschlüsselt zu werden – eine Abwägung von Kosten und Nutzen.

Bei der symmetrischen Verschlüsselung müssen Sender und Empfänger einer verschlüsselten Nachricht den geheimen Schlüssel (das Passwort) kennen.

Wenn der Nachrichtenaustausch nur zwei Teilnehmer betrifft, ist dies kein Problem. Aber stellen Sie sich ein System mit 100 Teilnehmern vor, in dem Benutzer mit jedem anderen auf geheime Weise kommunizieren soll. Wenn das System nur einen einzigen Schlüssel verwenden würde, könnte der Benutzer Joe nicht sicherstellen, dass die Nachricht tatsächlich von Benutzer Jane verschickt wurde. Dazu bräuchte jeder Benutzer einen eigenen Schlüssel. Und jeder Benutzer müsste die Schlüssel aller anderen Benutzer kennen. 99 Schlüssel zu verwalten und sich möglicherweise auch noch zu merken – das klingt ganz und gar nicht lustig.

Die Hauptprobleme der Verschlüsselung mit geheimem Schlüssel bestehen darin, dass die Anzahl der Schlüssel mit der Anzahl der Benutzer im System steigen, und dass jeder Benutzer so viele Schlüssel kennen muss, wie es Benutzer gibt.

Verschlüsselung mit öffentlichem Schlüssel

Betrachten wir noch einmal das System mit den 100 Benutzern, das wir im vorherigen Abschnitt angesprochen haben. Statt dass 99 seinen geheimen Schlüssel kennen müssen, stellt Joe einen öffentlichen Schlüssel zur Verfügung und behält einen privaten geheimen Schlüssel für sich. Jeder der 99 anderen Benutzer kann nun den öffentlichen Schlüssel verwenden, um eine Nachricht zu verschlüsseln und diese an Joe zu schicken. Nur Joe kann sie mit seinem privatem Schlüssel entschlüsseln. Dieses System hat einen offensichtlichen Mangel: Wir haben keine Berechtigung mehr, Joe weiß nicht, wer ihm die Nachricht geschickt hat, da jeder Benutzer die Nachricht verschlüsselt haben könnte. Der Absender muss daher mit seinem privaten Schlüssel unterzeichnen, so dass der Empfänger mit dem öffentlichen Schlüssel des Senders abgleichen kann, um die Authentizität und Integrität der Daten sicherzustellen. Dieses System wird als Verschlüsselung mit öffentlichem Schlüssel bezeichnet. Die beiden bekanntesten Algorithmen hierfür sind Diffie-Hellman und RSA (RSA steht für die Erfinder des RSA-Verschlüsselungsystems Rivest, Shamir und Adleman).

Der Hauptvorteil der Verschlüsselung mit öffentlichem Schlüssel gegenüber der Verschlüsselung mit geheimem Schlüssel besteht in dem höherem Niveau an Sicherheit und Benutzerfreundlichkeit. Es müssen keine privaten Schlüssel an andere Teilnehmer weitergeleitet werden. Im Gegenteil – bei der Verschlüsselung mit geheimem Schlüssel muss der geheime Schlüssel über einen Kommunikationskanal ausgetauscht werden. Dies bietet für Hacker wieder die Möglichkeit, durch Abhören während der Übertragung den Schlüssel herauszubekommen.

Ein weiterer Vorteil ist, dass Systeme mit öffentlichen Schlüsseln digitale Unterschriften bereitstellen können, indem ein Benutzer die Nachricht mit seinem privaten Schlüssel unterschreibt. Die Verschlüsselung mit geheimen

Schlüsseln erfordert dagegen eine zentrale Datenbank mit Kopien aller geheimen Schlüssel eines Systems, damit digitale Signaturen möglich sind. Kerberos verwendet dieses Verfahren. Ein zentrales Thema bei kritischen Daten bleibt jedoch das potentielle Risiko.

Ein Nachteil ist möglicherweise die Leistung. Viele Algorithmen mit geheimen Schlüsseln sind bedeutend schneller als Systeme mit öffentlichen Schlüsseln.

Die Verschlüsselung mit öffentlichen Schlüsseln soll die Verschlüsselung mit geheimen Schlüsseln nicht ersetzen. In einigen Situationen ist die Verschlüsselung mit öffentlichen Schlüsseln unnötig, und nur die Verschlüsselung mit geheimen Schlüsseln reicht aus. Wenn Sie Daten auf dem Server speichern, verwenden Sie wahrscheinlich eine Verschlüsselung mit einem einzigen Schlüssel. Da es keine unterschiedlichen Benutzer in diesem Szenario gibt und das System den Schlüssel zur Verschlüsselung und Entschlüsselung kennt, bringt es keinen Vorteil, einen öffentlichen und einen privaten Schlüssel zu haben. Für die Übertragung von Daten zu einem entfernten System (z.B. beim Verschicken von Bestellungen von einem Onlineshop via E-Mail) wird die Verschlüsselung mit öffentlichen Schlüsseln vorgezogen, da Sender und Empfänger zwei unterschiedliche Benutzer sind, die über einen unsicheren Kanal kommunizieren.

Der Verschlüsselungsstandard: Pretty Good Privacy (PGP)

Leider bietet PHP noch keine Unterstützung für PGP (Pretty Good Privacy). Da es jedoch bereits einige Open-Source-Alternativen gibt (z.B. Gnu Privacy Guard `www.gnupg.org`), sind wir überzeugt, dass dies nur noch eine Frage der Zeit ist. Zwischenzeitlich haben wir die in Listing 4.4 aufgeführte Basisklasse entwickelt, um eine Befehlszeilenversion von PGP nutzen zu können. Diese Klasse ermöglicht Ihnen, Dateien oder Zeichenketten mit PGP 6.5.1 zu verschlüsseln, zu entschlüsseln und zu signieren.

```
class pgp
{
    var $pgp_bin    = "/usr/bin/pgp";  // Path to PGP binary
    var $tmp_path   = "/tmp";          // Path where temporary files are
                                          ➥stored
    var $error;                        // Used to store the last error
                                          ➥message

    function pgp()
    {
        // Check if the PGP binary exists
        if(!file_exists($this->pgp_bin))
        {
```

```
        $this->error = "PGP binary file ".$this->pgp_bin." does not
        ➥exist.\n";
        return(false);
    }

    // Check if the PGP binary is actually executable
    if(!is_executable($this->pgp_bin))
    {
        $this->error = "PGP binary file ".$this->pgp_bin." is not
        ➥executable.\n";
        return(false);
    }

    return(true);
}

function _check_file($file)
{
    if(!file_exists($file))
    {
        // Create a temporary filename in the path specified as
           ➥class variable
        $temp_file = tempnam($this->tmp_path, "PGP").".asc";

        // Gently touch the file
        touch($temp_file);

        // Open the newly created file, write the string passed as
           ➥argument  $file to it
        $fp = fopen($temp_file, "w");
        if (!$fp)
        {
            $this->error = "Could not open temporary file $temp_file
            ➥for  writing in _check_file().\n";
            return(false);
        }
        fputs($fp, $file);
        fclose($fp);

        // Assign the temporary filename to $file
        $file = $temp_file;
    }

return($file);
}
```

```
function _exec_pgp_command($args)
{
    // Create a temporary filename in the path specified as class
       ➥variable
    $temp_file = tempnam($this->tmp_path, "PGP").".asc";

    // Execute the PGP command
    $command = $this->pgp_bin." -o $temp_file $args ";
    exec($command);

    // Open the temporary file created by PGP and read it into
       ➥$contents
    $fp = fopen($temp_file, "r");
    if (!$fp)
    {
        $this->error = "Could not open temporary file $temp_file for
        ➥reading in _exec_pgp_command().\n";
        return(false);
    }
    $contents = fread($fp, filesize($temp_file));
    fclose($fp);

    // Delete the temporary file
    unlink($temp_file);

    // Return the encrypted contents
    return($contents);
}

function encrypt($file, $my_user_id, $to_user_id)
{
    $file = $this->_check_file($file);
    $ret =  $this->_exec_pgp_command("-e -u \"$my_user_id\" -a
                    ➥\"$file\"  \"$to_user_id\"");

    return($ret);
}

function sign($file, $my_user_id)
{
    $file = $this->_check_file($file);
    $ret =  $this->_exec_pgp_command("-s -a -u \"$my_user_id\"
    ➥$file");

    return($ret);
}
```

```
    function encrypt_sign($file, $my_user_id, $to_user_id)
    {
        $file = $this->_check_file($file);
        $ret =  $this->_exec_pgp_command("-es -a -u $my_user_id $file
        ➥$to_user_id");

        return($ret);
    }

    function encrypt_conventional($file, $passphrase)
    {
        $file = $this->_check_file($file);
        $ret =  $this->_exec_pgp_command("-c -a -z \"$passphrase\"
        ➥$file");

        return($ret);
    }

    function decrypt($file, $my_user_id)
    {
        $file = $this->_check_file($file);
        $ret =  $this->_exec_pgp_command("-c $file -u \"$my_user_id\"");

        return($ret);
    }

    function decrypt_conventional($file, $passphrase)
    {
        $file = $this->_check_file($file);
        $ret =  $this->_exec_pgp_command("-z \"$passphrase\" $file");

        return($ret);
    }
}
```

Listing 4.4: PHP-Schnittstelle zu PGP 6.5.1

Da die Klasse pgp die PGP-Binärwerte Ihres Systems nur mit den entsprechenden Argumenten aufrufen, brauchen Sie ein richtig konfiguriertes PGP-System. Insbesondere Ihr privater Schlüssel muss richtig eingestellt sein, und alle öffentlichen Schlüssel, für die Sie eine Verschlüsselung vornehmen wollen, müssen sich in Ihrem lokalen »Schlüsselbund« befinden. Der öffentliche Schlüssel muss ein zuverlässiger Schlüssel sein; ansonsten fragt PGP, ob es den Schlüssel für diesen Benutzer verschlüsseln kann, und die Klasse schlägt fehl.

Alle Funktionen arbeiten entweder im Rahmen einer Datei oder einer Zeichenkette. Wenn Sie eine Zeichenkette übergeben, wird diese als temporäre Datei in `$tmp_path` gespeichert, da PGP nur mit Dateien arbeitet.

> **Warnung:** In einem Mehrbenutzersystem muss jeder diese Datei lesen können! Sie sollten sorgfältig abwägen, ob Sie diese Klasse in einem nichtzuverlässigen System verwenden (d.h. nicht zuverlässigen Benutzern Zugriff auf dieses System gewähren).

Die Klasse hat sechs »öffentliche« Funktionen; zwei weitere werden intern verwendet. Bei Auftreten eines Fehlers geben diese Funktionen `false` zurück. In diesem Fall können Sie über `$pgp->error` auf eine ausführliche Fehlermeldung zugreifen.

- `void pgp()`
  Der Konstruktor dieser Klasse überprüft ob auf die PGP-Binärwerte zugegriffen werden kann. Wenn ja, wird `true` zurückgegeben, ansonsten `false`.
- `mixed encrypt`(String `what`, String `my_user_id`, String `to_user_id`)
  PGP verschlüsselt das Argument `what`, das ein Dateiname oder eine Zeichenfolge sein kann, mit dem privaten Schlüssel von `my_user_id` für den öffentlichen Schlüssel `to_user_id`. Zurückgegeben wird der verschlüsselte Text oder bei einem Fehler `false`.
- `mixed sign`(String `what`, String `my_user_id`)
  Unterzeichnet das Argument `what` mit dem privaten Schlüssel von `my_user_id`. Zurückgegeben wird der unterzeichnete Text oder bei einem Fehler `false`.
- `mixed encrypt_sign`(String `what`, String `my_user_id`, String `to_user_id`)
  Unterzeichnet `what` mit dem privaten Schlüssel von `my_user_id` und verschlüsselt es dann für den öffentlichen Schlüssel von `to_user_id`. Zurückgegeben wird der unterzeichnete und verschlüsselte Text oder bei einem Fehler `false`.
- `mixed encrypt_conventional`(String `what`, String `passphrase`)
  Verschlüsselt `what` nur mit der konventionellen Verschlüsselung, wobei `passphrase` als geheimer Schlüssel verwendet wird. Zurückgegeben wird der verschlüsselte Text oder bei einem Fehler `false`.
- `mixed decrypt`(String `what`, String `my_user_id`)
  Entschlüsselt `what` mit `my_user_id` als privaten Schlüssel. Zurückgegeben wird der entschlüsselte Text oder bei einem Fehler `false`.
- `mixed decrypt_conventional`(String `what`, String `passphrase`)
  Entschlüsselt `what` mit der konventionellen Entschlüsselung, wobei `passphrase` als geheimer Schlüssel verwendet wird. Zurückgegeben wird der entschlüsselte Text oder bei einem Fehler `false`.

Die Mcrypt-Funktionen

Mit der Bibliothek MCrypt sind viele Blockalgorithmen verfügbar, einschließlich DES, TripleDES, Blowfish und IDEA. Der Platz reicht nicht aus, um all diese Algorithmen zu erklären oder Empfehlungen auszusprechen, welche Sie für ein bestimmtes Szenario verwenden sollten. Die Systeme werden ausführlich in vielen Fachbüchern und Online-Artikeln behandelt, von denen einige im Abschnitt »Ressources« auf der CD-ROM aufgeführt sind.

Wie bereits in Kapitel 1 »Entwicklungskonzepte« erwähnt, bringt eine weitere Bibliothek leider auch einen weiteren API-Stil mit sich, was wir nicht für erstrebenswert halten. Warum hat `mcrypt_cbc()` ein Argument, das definiert, ob Daten verschlüsselt oder entschlüsselt werden? Wäre es nicht logischer und konsistenter, dafür zwei Funktionen `mcrypt_encrypt_cbc()` und `mcrypt_decrypt_cbc()` zu verwenden? Es gibt auch kein `session_var()` mit dem Argument `REGISTER` oder `UNREGISTER`.

Genug der Nörgelei. Immerhin könnten wir einfach die Quelle für die Mcrypt-Schnittstelle bearbeiten und diese zusätzlichen Funktionen definieren. Dies ist der Vorteil einer Open-Source-Software. Zurück zum Thema. Das Beispiel in Listing 4.5 zeigt die Mcrypt-Funktion in Gebrauch. Das Beispiel durchläuft mehrfach ein Array, das alle möglichen Mcrypt-Algorithmen enthält, und verschlüsselt mit jedem Algorithmus eine Nachricht.

```
// Set up an array of algorithms generally supported by Mcrypt
$algorithms = array(
        MCRYPT_BLOWFISH,
        MCRYPT_DES,
        MCRYPT_TripleDES,
        MCRYPT_ThreeWAY,
        MCRYPT_GOST,
        MCRYPT_CRYPT,
        MCRYPT_DES_COMPAT,
        MCRYPT_SAFER64,
        MCRYPT_SAFER128,
        MCRYPT_CAST128,
        MCRYPT_TEAN,
        MCRYPT_RC2,
        MCRYPT_TWOFISH,
        MCRYPT_TWOFISH128,
        MCRYPT_TWOFISH192,
        MCRYPT_TWOFISH256,
        MCRYPT_RC6,
        MCRYPT_IDEA
        );

$message = "Hello PHP world.";  // Message to be encrypted
```

```
$secret = "Secret password";     // Secret key

for($i=0; $i<count($algorithms); $i++)
{
    // If this algorithm is available, $algorithms[$i] is an integer
       ➥constant
    if (is_integer($algorithms[$i]))
    {
        print("<b>$algorithms[$i]:
        ➥".mcrypt_get_cipher_name($algorithms[$i]).":</b><br>");
    }
    else
    {
        print("<b>$algorithms[$i] is not supported</b><br>");
        continue;
    }
    // Get the block size of the current algorithm
    $block_size = mcrypt_get_block_size($algorithms[$i]);

    // Create an initialization vector from device /dev/random
    $iv = mcrypt_create_iv($block_size, MCRYPT_DEV_RANDOM);

    // Encrypt the plaintext with $algorithms[$i]
    $encrypted = mcrypt_cbc($algorithms[$i], $secret, $message,
    ➥MCRYPT_ENCRYPT, $iv);

    // Decrypt it again
    $unencrypted = mcrypt_cbc($algorithms[$i], $secret, $encrypted,
    ➥MCRYPT_DECRYPT, $iv);

    // Output plaintext and ciphertext
    print("Ciphertext: $encrypted<br>");
    print("Plaintext: $unencrypted<p>");
}
```

Listing 4.5: Mcrypt-Routinen

MCrypt verwendet zur Verschlüsselung und Entschlüsselung von Daten Blockverschlüsselungsalgorithmen. Die Blockverschlüsselung ist eine Anwendung der bereits erwähnten symmetrischen Verschlüsselungsmethode (Gegenteil: Verschlüsselung mit öffentlichem Schlüssel). Wenn Sie eine Nachricht mit einer Blockverschlüsselung bearbeiten, können Sie verschiedene Betriebsarten verwenden, um den Klartext zu verschlüsseln. ISO92b definiert vier generische Modi, die für die Verschlüsselung von Blöcken beliebiger Größe eingesetzt werden können: Electronic Code Book, Cipher Block Chaining, Cipher Feedback und Output Feedback.

- *Electronic Code Book (ECB)*- Modus sollten Sie nur mit äußerster Vorsicht verwenden. Jeder 64-Bit-Block des Klartextes wird nacheinander mit den angegebenen Algorithmen unabhängig voneinander verschlüsselt. Die Klartextmuster werden nicht verborgen. Sie sind als Iterationen im verschlüsselten Text zu sehen. ECB eignet sich daher nur für die Verschlüsselung von willkürlichen Daten, z.B. einem MD5-Hash.
- *Cipher Block Chaining (CBC)* vermeidet dieses Problem, indem jeder Block vor der Verschlüsselung durch XOR mit dem vorangegangenen verknüpft wird. Die Verschlüsselung eines Blocks hängt demnach von den vorhergehenden Blöcken ab. Derselbe 64-Bit-Klartextblock kann je nach seiner Position in der Nachricht zu unterschiedlich verschlüsselten Blöcken führen. Da es als willkürlichen Startparameter für XOR auch einen Initialisierungsvektor benutzen kann, ist es sehr viel sicherer als ECB.
- *Cipher Feedback (CFB)*- Modus macht es genau andersherum: Der Klartext wird verschlüsselt und anschließend durch XOR mit dem vorhergehenden Klartextblock verknüpft. Der Vorteil von CFB ist, dass es mit Blöcken kleiner als 64-Bits arbeitet und damit auch zur Verschlüsselung von Byte-Strömen verwendet werden kann.
- *Output Feedback (OFB)*- Modus ähnelt dem CFB-Modus, hat jedoch einen Vorteil: Bit-Fehler, die möglicherweise während der Übertragung auftreten, beeinflussen nicht die Entschlüsselung von benachbarten Blöcken. Durch Ändern des verschlüsselten Textes kann der Klartext, genau wie beim ECB-Modus, leicht verändert werden.

In der Praxis findet CBC die größte Verbreitung. TripleDES mit CBC-Modus bietet Ihnen soviel Sicherheit, wie Sie jemals in Ihren Webanwendungen brauchen werden.

4.2.3 Integrieren Sie Fachleute in Ihr Team

Wenn Sicherheit in Ihrem Projekt zu einem wesentlichen Faktor wird, ist es hilfreich, qualifiziertes Personal für die Qualitätssicherung und die Sicherheitsüberprüfung zu haben. Selbst wenn Sie ein sehr erfahrener Programmierer sind, bleiben Sie doch menschlich – und als solcher können Sie irren. Mit einem zweiten professionellen Blick über Ihre Schulter bei wichtigen Teilen eines Programms erhalten Sie die nötige Doppelkontrolle.

Leider ist es sehr schwierig, qualifiziertes und nicht zu teures Personal mit Kenntnissen in Programmsicherheit zu finden. Dieses Thema wird zunehmend an den Universitäten gelehrt, aber die beste Ausbildung kann fehlende Praxiserfahrung nicht ersetzen. Dies gilt für alle Berufe im Bereich neue Medien, aber ganz besonders für den Bereich der Computer- und Softwaresicherheit. Diese Schwierigkeiten lassen sich überwinden, indem Sie Ihre Soft-

ware mit einer Open-Source-Lizenz freigeben. Möglicherweise werden dann die Industrieexperten selbst Ihr System prüfen. Open-Source mit seinem offenen Entwicklungsprozess erfordert wirkliche Sicherheit – nicht Sicherheit durch Verschleierung, sondern Sicherheit gegen alle möglichen Attacken durch einen Hacker, der sich sehr gut mit dem System und seinem Quellcode auskennt.

Selbst wenn Open-Source für Sie keine Möglichkeit ist, können Sie viele der Programmentwicklungsprinzipien, die in der Open-Source-Gemeinde verwendet werden, auch für Ihr eigenes Produkt einsetzen. Eine kontinuierliche Kontrolle durch Experten kann beispielsweise helfen, die üblichen Sicherheitsfehler zu vermeiden, die durch unzureichendes Testen entstehen.

Diese Kontrolle kann in Form von regelmäßigen Überprüfungen des Programmcodes formalisiert werden. Damit werden nicht nur sicherheitsbezogene Mängel, sondern auch allgemeinere Softwarefehler vermieden. In wöchentlichen Sitzungen geht ein Team von bis zu fünf Personen den neuen Quellcode durch und überprüft ihn auf bekannte Fehler, wie etwa fehlende Bereichsüberprüfungen, nicht verwendete Variablen oder Sicherheitsfragen. Ein Moderator zeichnet alle gefundenen Fehler auf und stellt sicher, dass sie später vom ursprünglichen Entwickler korrigiert werden. Ein typisches Testszenario könnte folgendermaßen aufgebaut sein:

1. Ein Team von zwei bis fünf Entwickler wird gebildet.
2. Etwa 60 Minuten lang wird der Quellcode (etwa 200 bis 300 Zeilen Programmcode) überprüft. Jedes Teammitglied geht den Programmcode Zeile für Zeile durch und versucht zu verstehen, was jede einzelne Zeile tut. Alle gefundenen Fehler werden notiert.
3. Der Moderator sammelt die Notizen ein und leitet sie an den ursprünglichen Programmentwickler weiter, der nicht Teil des Prüfteams sein sollte. Die Überarbeitung kann daraus bestehen, dass der Programmcode geändert wird, Kommentare hinzugefügt oder gelöscht werden, Funktionen neu strukturiert oder umgebettet werden usw.

Wir bezeichnen dies als die einfache Software-Inspektion (Discount Software Inspection). Bei der traditionellen Softwareentwicklung nimmt die Überprüfung einen sehr viel größeren Raum ein. Fagan Inspection, das ursprünglich von Michael Fagan bei IBM 1976 [3] entwickelt wurde, benötigt größere Vorbereitung, ausführlichere Prüflisten und häufigere Meetings. Die Entwicklung von Webanwendungen ähnelt der raschen Entwicklung von Prototypen, da eine schnelle Markteinführung von wesentlicher Bedeutung ist. Prüfverfahren die zu zeitaufwendig und ressourcenintensiv sind, haben für die Praxis keine Bedeutung.

4.2.4 Authentisierung

Sie werden bald feststellen, dass Sie eine Technik immer wieder brauchen: Authentisierung. Wenn Ihre Benutzer beispielsweise nur dann auf einige Teile Ihrer Website zugreifen dürfen, nachdem sie sich selbst im System eingetragen haben, brauchen Sie ein Authentisierungsverfahren.

Der Anmeldevorgang

Betrachten wir zunächst die theoretischen Konzepte der Genehmigung. Die Anmeldung findet in drei Stufen statt: Identifizierung, Berechtigung und Genehmigung.

Identifizierung

Bevor Sie überprüfen können ob ein Benutzer berechtigt ist, müssen Sie seine Identität kennen. Daher fordern Sie ihn auf, sich zu identifizieren. Die Identifizierung gibt an, wer der Benutzer ist. Sie kann ein Benutzername, eine Kundennummer oder etwas beliebig Anderes sein, solange Sie in Ihrer Benutzerbasis eindeutig ist. Der Begriff »Benutzer« kann sich in diesem Kontext auf eine Person, einen Prozess oder ein System (beispielsweise einen Knoten im Netzwerk) beziehen.

Identifizierung ist nicht dasselbe wie Authentisierung! Die Tatsache, dass ein Benutzer eine Identität angegeben hat, bedeutet nicht, dass diese Identität auch garantiert berechtigt ist. Ohne Authentisierung ist diese Identität suspekt.

Authentisierung

Nachdem der Benutzer eine Identifizierung angegeben hat, müssen Sie diese überprüfen. Dies ist die Aufgabe eines Authentisierungssystems. Es überprüft die Identität des Benutzers mit einem von drei Authentisierungsverfahren: etwas, das sie wissen, etwas, das sie haben oder etwas, das sie sind.

Das Verfahren »etwas, das sie wissen«, das auch als Authentisierung durch Wissen bezeichnet wird, ist die im Web am häufigsten verwendete Methode. Benutzer können ein Passwort wählen oder sich zuweisen lassen. Dieses müssen sie sich merken und geheim halten. Die Vergabe der Authentisierung erfolgt durch Überprüfung, ob die Identität des Benutzers durch das Passwort bestätigt wird. Varianten sind die Personal Identification Numbers (PINs), die Übergabe von Phrasen oder die Anfrage nach Daten über den Benutzer, die nur der Benutzer wissen kann. Die Hauptschwäche dieser Art von Authentisierung besteht darin, dass es häufig recht einfach ist zu erfahren, was jemand anderer weiß. Möglicherweise können Sie sogar das magische »etwas, das sie wissen« erraten, ohne Zugriff darauf zu haben. Denken Sie an die Brechstangenattacken bei Shell-Logins. Im Falle von Webanwendungen überwiegen die Vorteile der Authentisierung durch Wissen normalerweise diese Sicherheitsmängel. Der Benutzer kann das Passwort überall ablegen und ständig auf dieses zugreifen. Ein weiterer Vorteil ist, dass diese Methode sehr einfach ist und sich leicht installieren lässt.

Beispiele für die Authentisierung durch Eigentum (etwas, das sie haben) sind Schlüssel, Magnetstreifenkarten, Kennmarken u.ä. Im Gegensatz zur ersten Methode ist die Authentisierung durch Eigentum schwieriger zu duplizieren, da die Authentisierungselemente physikalische Objekte sind. Für das Web hat sich diese Art der Berechtigung noch nicht als gültige Technik etabliert. Es gibt jedoch bereits einige Bemühungen, Magnetstreifenkarten für Verschlüsselungssysteme mit öffentlichen Schlüsseln einzuführen.

Die dritte Art der Authentisierung ist sehr viel gebräuchlicher. Die Authentisierung nach Merkmalen (etwas, das sie sind) wird beispielsweise in Firewall-Systemen verwendet, um einem Objekt nur zu den Systemen mit einer bestimmten IP-Adresse oder in einem bestimmten Bereich Zugriff zu gewähren. Außerhalb dieses Web-Bereichs wird die Authentisierung nach Merkmalen vermehrt in Systemen eingesetzt, die Retina-Abtastungen oder Fingerabdrücke als Berechtigung verwenden. Dieses Verfahren ist zwar das sicherste der drei (immerhin zielt die Berechtigung darauf ab, sicherzustellen wer Sie sind, und dieser dritte Typ kommt diesem Ziel sehr nah. Seine Umsetzung ist aber auch die kostenintensivste. Wie wir bereits betont haben, können IP-Adressen zur Identifizierung von Einzelpersonen nicht in Betracht gezogen werden. Sie brauchen immer noch ein persönliches Identifizierungssystem, das ein Peripheriegerät verwendet.

Die unterschiedlichen Authentisierungsverfahren lassen sich natürlich kombinieren, damit das Ergebnis sicherer wird. Vorläufig möchten wir jedoch unsere Anmeldeprozedur mit dem Authentisierungsverfahren nach Wissen fortsetzen.

Der Benutzer gibt zusammen mit seiner Identität ein Authentisierungselement ein, das häufig ein Passwort ist. Doch wie werden diese Daten vom Client zum Authentisierungssystem weitergeleitet? Die Identifizierungsdaten sind sehr anfällig für einen Eingriff eines Eindringlings, der zwischen dem Benutzer und dem Authentisierungssystem sitzt (Mann-in-der-Mitte-Attacke). Folglich müssen Sie sich gegen die Lauschangriffe schützen. Ein zuverlässiger Pfad – ein sicherer Übertragungsweg – ist notwendig, um das Passwort zu übertragen. Bedenken Sie, dass eine Authentisierungskette nur so sicher ist, wie ihr schwächstes Glied. Selbst ein 128-Bit langes Kennwort hilft Ihnen nicht, wenn sie es über eine normale HTTP-Verbindung an das Authentisierungsystem schicken.

Der Datenverkehr sollte sicher sein

Selbst wenn Sie die Identifizierungsdaten über einen sicheren Übertragungskanal wie SSL weiterleiten, bleibt noch sehr viel Raum für Mann-in-der-Mitte-Attacken, wenn der Datenverkehr nach der Authentisierung nicht verschlüsselt wird. Ein Angreifer erfährt möglicherweise die Sitzungskennnummer durch Lauschen und kann so effektiv die Sitzung des Benutzers kapern und die Identität des Benutzers übernehmen (z.B. Artikel in einem Online-Shop bestellen).

Um dies zu vermeiden, müsste der gesamte Datenverkehr über einen sicheren Kanal abgewickelt werden.

Das Authentisierungselement wird mit einer Authentisierungsdatenbank abgeglichen. Das System, das die Datenbank enthält, und die Art, in der die Authentisierungselemente gespeichert werden, müssen sicher und zuverlässig sein. Die Authentisierungsdatenbank benötigt einen Schutz für den allgemeinen Zugriff, und die Authentisierungselemente sollten verschlüsselt gespeichert werden.

Auch die Backup-Systeme müssen sicher und zuverlässig sein. Zu einem entsprechenden Trust Management gehört auch das Einrichten von Rollen und das Definieren, wer auf spezielle Teile eines zuverlässigen System in welcher Weise zugreifen kann. Trust Management ist jedoch ein zu großes Gebiet, als dass wir es hier abhandeln könnten.

Vergabe von Zugriffsrechten

Wenn die Benutzeridentifizierung laut Authentisierung korrekt ist, beendet das System den Anmeldevorgang und weist der Identität und den Zugriffskontrolldaten des Benutzers eine Benutzersitzung zu. In einfachen Programmen besteht die Zugriffskontrollinformation vielleicht nur aus einem Flag, das angibt, dass der aktuelle Benutzer berechtigt ist. In komplexeren Situationen ordnet das System möglicherweise auch eine Sicherheitsebene oder Authentisierungsebene zu, welche definiert, was der aktuelle Benutzer im Programm tun darf (z.B. gibt es vielleicht einen übergeordneten Benutzer oder eine Nur-Lese-Benutzergruppe). Je nach benötigter Sicherheitsebene muss das System erfolgreiche oder fehlgeschlagene Anmeldeversuche protokollieren. So erlegt der C2-Sicherheitsstandard dem System auf, alle Anmeldeereignisse aufzuzeichnen.

HTTP-Authentisierung

HTTP bietet eine Methode für die Benutzer -Authentisierung: die HTTP Basic Authentication. Auf Seiten, die eine Authentisierung vorschreiben, antwortet der Webserver dem Client mit einem bestimmten Kopf:

```
HTTP/1.1 401 Authorization Required
WWW-Authenticate: Basic realm="Protected Area"
```

Daraufhin erscheint im Browser ein modaler Dialog, der den Benutzernamen das Passwort verlangt. Dieser Berechtigungstyp ist jedoch wirklich nur als »grundlegend« zu bezeichnen, denn es beinhaltet eine Reihe von Nachteilen:

- Um Benutzer abzumelden, müssen Sie Tricks anwenden.
- Um Benutzer nach einer definierten Leerlaufzeit abzumelden, müssen Sie noch mehr Tricks anwenden.

- Wenn Sie Benutzergruppen benötigen (die wir zuvor als »Genehmigungsebenen« bezeichneten), brauchen Sie eine eigene Programmlogik, um Einzelbenutzer zu Gruppen zusammenzufassen.
- Sie können den Anmeldeprozess nicht verändern: Der Popup-Dialog bleibt immer derselbe, egal welche Website Sie verwenden.
- Unerfahrene Benutzer haben in der Regel Angst vor diesen Dialogen. Sie können keine Hilfe bieten, da Sie den Dialog nicht verändern können.
- HTTP-Authentisierung über PHP ist nur mit der Modulversion möglich.
- Die Berechtigung beschränkt sich auf Verzeichnisse. Was ist, wenn Sie nur eine einzige Seite schützen möchten? Mit den Apache-Anweisungen ist dies unmöglich. Sie müssten eine HTTP-Authentisierung über PHP durchführen.

All dies führt zur Schlussfolgerung, dass es möglicherweise klüger ist, eine andere Lösung zu bevorzugen, wenn Sie es nicht gerade mit sehr einfachen Szenarien zu tun haben, bei denen Sie wissen, dass Ihr Publikum diese Dialoge kennt und Sie keine Genehmigungsebenen oder Leerlauf-Timeouts benötigen.

PHP-Authentisierung

Die PHP-eigene Authentisierung ermöglicht Ihnen dagegen, beliebige Anmeldemasken und Berechtigungsprozeduren zu verwenden, da sie formularbasiert arbeitet. Damit die Berechtigung funktioniert, brauchen Sie auch Sitzungsverwaltungsfunktionen. Sobald der Benutzer angemeldet ist, müssen Sie sich diesen Status über mehrere Anfragen hinweg merken.

Sie können sich eine eigene Authentisierungsbibliothek schreiben, wenn Sie dafür die zuvor besprochenen PHP-Sitzungsverwaltungsfunktionen verwenden. Oder Sie benutzen PHPLib. Mit seiner Klasse *Auth* können Sie die Authentisierung durchführen und mit der Klasse *Perm* können Sie alle Zugriffsrechte vergeben. Einzelheiten über die genaue Vorgehensweise finden Sie in Kapitel 6.

4.3 Die Bedeutung der Tauglichkeit

Sie wundern sich vielleicht, warum in einem Buch über Softwareentwicklung das Thema Tauglichkeit angeschnitten wird. Wir halten es für notwendig, dass ein ernsthafter Entwickler die grundlegenden Prinzipien für die Informationsarchitektur, der Entwicklung von Benutzerschnittstellen und der Tauglichkeit kennen.

Da Webanwendungen immer größer und komplexer werden, sind Webentwickler mehr denn je gefordert, effektive und funktionale Websites zu erstellen. Tauglichkeit wird ein Schlüsselbegriff für diese Websites.

Was ist Tauglichkeit? Die Eigenschaft, wie einfach ein Informationssystem zu erlernen und zu benutzen ist, bezeichnet man als Systemtauglichkeit. Als Programmentwickler fällt es Ihnen wahrscheinlich leicht, Ihr System zu benutzen. Aber Sie wären überrascht, wie schwierig es andere Benutzer finden. Zunächst sind sie mit dem neuen System absolut nicht vertraut und versuchen möglicherweise, es für andere Dinge zu verwenden, als Sie beabsichtigten!

Tauglichkeitsexperten haben versucht Eignungsfragen bereits in einem frühen Stadium der Softwareentwicklung einfließen zu lassen. Dies war jedoch nicht sehr erfolgreich. Die Eignung spielt jedoch mit Beginn jedes Projektes eine große Rolle. Diese Überlegung erst in der Beta-Testphase einzubringen, reicht nicht aus. Unserer Meinung nach sollte der Tauglichkeit dieselbe Bedeutung wie den traditionellen Merkmalen der Softwarequalität beigemessen werden, wie etwa Korrektheit, Pflege und Zuverlässigkeit. Sobald Ihnen als Softwareentwickler bewusst ist, wie wichtig die Tauglichkeit für die Sicherung der Programmqualität ist, werden Sie sich bemühen, die Erfahrung des Benutzers zu fördern. Wie Sie die Tauglichkeit Ihres Programms verbessern können, hängt vom einzelnen Projekt ab. Einige wichtige Prinzipien sind jedoch für die Entwicklung aller benutzerorientierten Programme gültig:

- Frühe Konzentration auf den Benutzer; direkter Einbezug desselben in die Konzeption
- Frühe und kontinuierliche Bewertung des Programms
- Empirische Messung der Tauglichkeit, selbst in frühen Phasen der Entwicklung
- Iterative Konzeption und Entwicklung

4.3.1 Tauglichkeit in Webanwendungen

Webanwendungen unterscheiden sich grundlegend von Desktop-Programmen.

Mit HTML können Sie das Layout nicht zu 100% zuverlässig kontrollieren. Sie müssen bei der Anzeige Kompromisse eingehen. Möglicherweise wird Ihre Website auf einer Vielzahl von Anzeigegeräten dargestellt. Dies kann von Palm-Pilots bis Web-TV mit einem Standardbrowser auf einem 800 x 600-Bildschirm variieren.

Die Benutzerinteraktion wir aufgrund der heute noch niedrigen Bandbreite stark gebremst. Auf der Client-Seite können Sie damit nur die einfachsten Basisskripte mit JavaSkript ausführen.

Bei der traditionellen Softwareentwicklung können Sie die Aktionen des Benutzers beeinflussen. Sie können Menüpunkte grau hinterlegen oder modale Dialogboxen ausgeben, welche das Programm solange blockieren, bis der Benutzer die Fragen beantwortet hat. Auf der Website können Sie jedoch nicht kontrollieren, wie sich der Benutzer bewegt. Er kann über direkte Links, Lesezeichen oder Suchmaschinen auf Ihre Website stoßen.

Sie können auch nicht erwarten, dass der Benutzer ein Handbuch liest, um sich mit Ihrer Webanwendung vertraut zu machen (wie es bei traditionellen Programmen üblich ist), denn Benutzer wechseln häufig sehr schnell von einer Site zur nächsten. Die Hypertext-Struktur verleitet den Benutzer häufig dazu, das Web als Ganzes und nicht als separates Programm oder einzelne Website zu benutzen.

Die grundlegenden Entwicklungsprinzipien aus »Usability 101« sind in bezug auf die Tauglichkeit nach wie vor gültig, und wir sollten auf einige der generischen Regeln eingehen. Die folgenden Richtlinien hängen voneinander ab. Sie sollten anhand Ihres spezifischen Programms entscheiden, welchen Rang Sie den einzelnen Regeln geben.

Eine benutzerfreundliche Webanwendung zeichnet sich durch folgende Merkmale aus:

- Sie eignet sich für die Aufgabe, die es durchführen soll.
- Sie lässt sich durch den Benutzer kontrollieren.
- Sie entspricht der Benutzererwartung.
- Sie ist benutzerspezifisch.
- Sie ist selbsterklärend.

Ist es geeignet?

Das Programm sollte dem Benutzer auf effektive und effiziente Weise helfen, sein Ziel zu erreichen. Benutzer interessieren sich in der Regel nicht für ausgefallene Grafiken; sie wollen lediglich ihr Ziel erreichen – egal ob dies nun der Erhalt von Informationen ist oder etwas Spezielleres, wie etwa das Bestellen eines Produktes. In einem Online-Shop sollte das System beispielsweise dem Benutzer helfen, den Bestellvorgang so mühelos wie möglich abzuwickeln. Amazon.com hat hierfür einen Standard entwickelt: Sobald sich ein Benutzer im System angemeldet hat, braucht er nur noch einen einzigen Klick, um ein Produkt zu bestellen. Eignung für eine bestimmte Aufgabe kann aber bereits bei einzelnen Dialogen beginnen:

- Zeigen Sie nur die Informationen, die der Benutzer zum Erreichen seines Ziels tatsächlich braucht.
- Stellen Sie Standardwerte zur Verfügung. Tragen Sie beispielsweise in einem Datumsfeld eines Formular bereits das aktuelle Datum ein.
- Zwingen Sie den Benutzer nicht, unnötige Schritte durchzuführen. Zeigen Sie beispielsweise Fehlermeldungen nicht auf einer neuen Seite an, so dass sich der Benutzer merken muss, welche Felder falsch ausgefüllte wurden und zur vorherigen Seite zurückkehren muss, sondern blenden Sie die Fehlermeldung direkt im Formular ein (über das PHP-Standardformular). Abbildung 4.2 stellt beide Ansätze gegenüber.

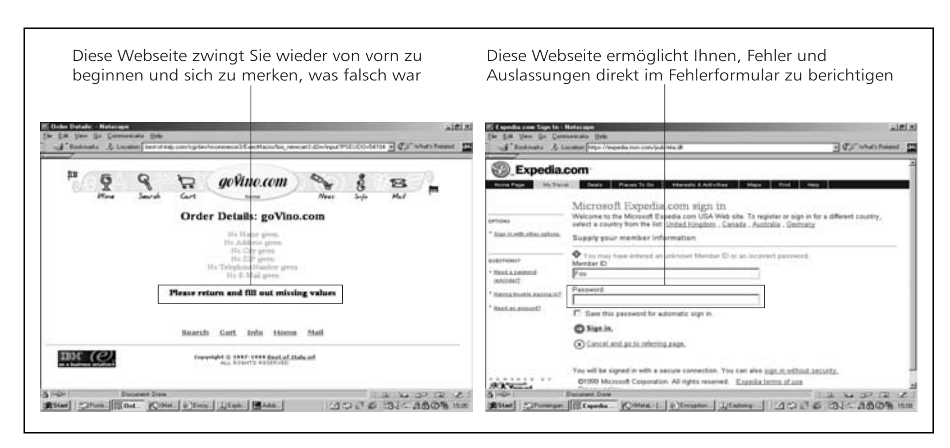

Abbildung 4.2: Stufen der Formularüberprüfung: Gute und schlechte Beispiele

Ist eine Benutzerkontrolle möglich?

Eine Webanwendung ist vom Benutzer kontrollierbar, wenn der Benutzer die Geschwindigkeit und Richtung des Programmablaufs beeinflussen kann, bis er sein Ziel erreicht.

Die Anzeigedauer einer Dialogbox sollte stets vom Benutzer kontrollierbar sein und nicht vom Programm diktiert werden. Dies klingt logisch. Uns ist aber auch ein Beispiel begegnet, wie Sie selbst diese grundlegende Regel brechen können. Nachdem Sie ein Formular falsch ausgefüllt haben, blendet das Programm eine Fehlermeldung ein und kehrt automatisch nach fünf Sekunden zum Formular zurück. Die korrekt eingegebenen Daten wurden natürlich nicht gespeichert, und Sie müssen von vorne beginnen (wenn es Ihnen das wert ist – wir haben diese Seite einfach verlassen). Benutzer, die nach dem Abschicken des Formulars zu einem anderen Programm oder einer anderen Browserinstanz gewechselt haben, hätten die Fehlermeldung sogar niemals gesehen.

Sie sollten dem Benutzer auch die Wahl lassen zu entscheiden, wie viele Daten angezeigt werden. Wenn sich ein Formular über mehrere Schritte erstreckt,

geben Sie ihm die Möglichkeit, zwischen den einzelnen Stufen hin und her zuschalten, ohne dass die Daten verloren gehen. Benutzer wählen häufig fälschlicherweise Funktionen und suchen dann einen Weg, diesen unerwünschten Zustand zu verlassen. Wenn das Programm vom Benutzer vorübergehend verlassen wird, ermöglichen Sie ihm, es später wieder zu betreten.

Kontrollierbar bedeutet auch, dass sich das Programm an die Bedürfnisse und Merkmale des Benutzers anpasst. In einem Intranet können Sie beispielsweise erfahrenere Benutzer erwarten, da sie sich mit dem System bei der täglichen Arbeit vertraut machen. Die Navigationsstruktur und das Programm sollten daher so gestaltet sein, dass der Benutzer über Tastaturkürzel oder andere Mittel wesentliche Punkte schneller erreicht, z.B. über Pulldown-Menüs, die einen direkten Sprung auf eine bestimmte Seite ermöglichen. Außerdem brauchen die Benutzer fortgeschrittenere Hilfeseiten, die das unterschiedliche Wissensniveau befriedigen.

Entspricht es der Benutzererwartung?

Einer der wichtigsten Aspekte für die Tauglichkeit ist Konsistenz. Jakob Nielsen, der wohl bekannteste Tauglichkeitsexperte, drückte es in seinem Gesetz zur Webbenutzererfahrung so aus: »Benutzer verbringen die meiste Zeit auf anderen Websites.« Die Konventionen, die sich auf den meisten anderen Websites etabliert haben, sollten daher auch auf Ihrer Website eingehalten werden. Dr. Nielsen's 14tägige Spalten mit Warnmeldungen geben so viel seiner Erfahrung und seines Wissens wieder, dass sie ein Muss für jeden sind, der im Bereich Produktentwicklung und Informationstechnologie arbeitet. In seiner Warnbox vom 22. August 1999 stellt Dr. Nielsen ein gutes Beispiel dafür vor, was passiert, wenn ein Programm den Erwartungen eines Benutzers nicht entspricht: [4]

Eric Davis, ein Informatiker bei Resource Marketing, berichtete kürzlich über einen Tauglichkeitstest für die Einkaufswagenterminologie. Das Konzeptdesign unterstützt den Begriff »Einkaufsschlitten«, da sich die Website (die sich auf den Verkauf von Wintersportprodukten bezog) von anderen abheben und die Standardterminologie vermeiden wollte. Ergebnis: 50% der Benutzer verstanden das Schlittenkonzept nicht. Die anderen 50% sagten, dass sie herausgefunden haben, was es bedeutete, weil es sich an demselben Standort befand, an dem auch ein Einkaufswagen sein würde. Sie wussten, dass sie irgendetwas hinzufügen mussten und das Einzige, was Sinn ergab, war der Schlitten. Lektion: Versuchen Sie nicht clever zu sein und verwenden Sie keine neuen Begriffe, wenn es schon gute Begriffe gibt, die der Benutzer bereits kennt.

Konsistenz bedeutet auch, dass das Verhalten von Dialogfenstern und die Datenanzeige darin einheitlich sein sollten:

- Zeigen Sie Systemmeldungen (Statusangabe, Meldungen über fehlgeschlagene oder erfolgreiche Aktionen) stets an derselben Stelle auf der Seite und in der gleichen Darstellung an.

- Verwenden Sie für die Benennung von Schaltflächen und Verknüpfungen ein konsistentes Namensschema.
- Verwenden Sie ein konsistentes Mittel, um den Status eines Dialogs zu ändern. Platzieren Sie beispielsweise die Schaltfläche zum Abschicken eines Formulars stets in die untere rechte Ecke.
- Erfinden Sie kein neues grafisches Benutzerelement, wenn es nicht notwendig ist. Eine Website, die wir überprüften, verwendete beispielsweise Bilder anstelle von reinen HTML-Kontrollboxen. Beim Klicken auf ein Bild wurde die gesamte Seite mit einer leichten Änderung des Kontrollkästchenbildes neu geladen (je nach vorherigem Status entweder aktiviert oder deaktiviert). Ein lästigeres Verfahren gibt es wohl kaum.

Ist es benutzerspezifisch?

»Personenspezifische Inhalte« – das ist eines der Schlagworte im Internet. Wir meinen damit nicht dasselbe wie die Marketingabteilung, die personenspezifische Werbung, personenspezifische Spammails oder personenspezifische Nachrichten anpreist. Für uns bezieht sich der Begriff »personenspezifische Webanwendungen« einfach auf Programme, die auf die Bedürfnisse und die kulturellen Eigenschaften individueller Benutzer zugeschnitten sind.

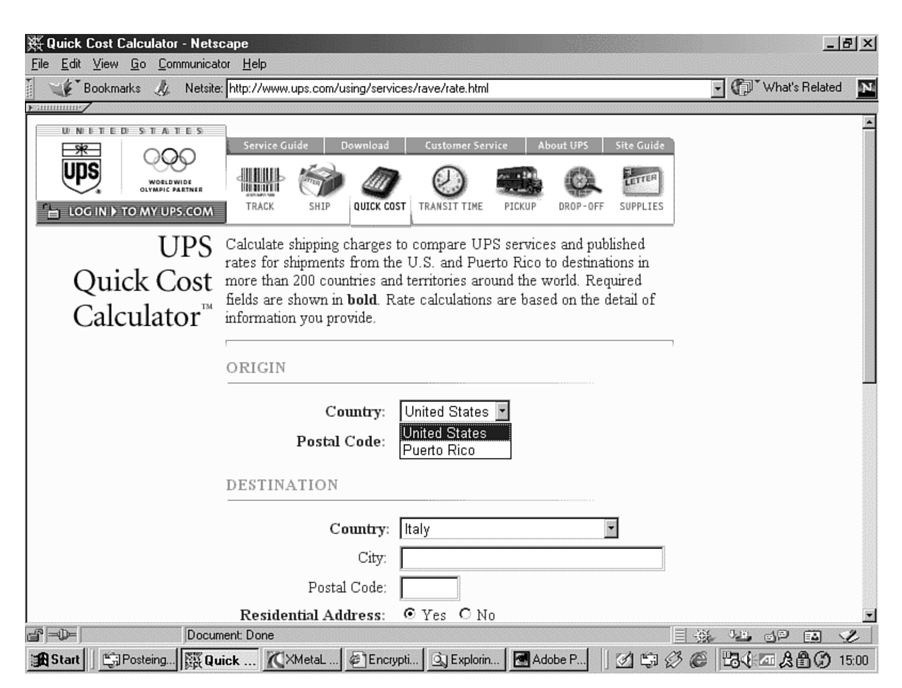

Abbildung 4.3: Das UPS-Versandkostenberechnungsprogramm akzeptiert als Ursprungsland für ein Paket nur die USA und Puerto Rico, auch wenn UPS aus über 50 Ländern Pakete verschickt.

Als europäischer Internetbenutzer sehen wir häufig das Vorurteil bestätigt, dass Amerikaner dazu tendieren, eine amerika-bezogene Sicht der Welt einzunehmen. Benutzernamen, die sich an deutschen Umlauten verschlucken, und Ablaufsteuerungsprogramme, in denen Sie keine Zeitzonendaten eingeben können, Informationssysteme, die auf amerikanischen ZIP-Codes basieren – die Liste ließe sich endlos fortsetzen (siehe Abbildung 4.3). Es erfordert zwar möglicherweise einen höheren Kosten- und Zeitaufwand, um ein Programm zu entwickeln, das von einem internationalen Publikum benutzt werden kann, aber es macht sich bezahlt, denn bereits im September 1999 kamen bereits 50% der Webbenutzer nicht aus den Vereinigten Staaten.

Ist es selbsterklärend?

Ein Programm ist selbsterklärend, wenn es dem Benutzer hilft, ein System zu erlernen und zu verstehen. Dies ist im Web besonders wichtig, da die meisten Benutzer in Hinsicht auf Ihre Website Laien sind. Sie wandern von einer Website zur nächsten und erinnern sich in der Regel nicht an die Konventionen und Regeln Ihrer Website, wenn sie diese das nächste Mal besuchen. Daher sollten Sie eher Zeit darauf verwenden, die Dialogfenster zu vereinfachen, anstatt ein ausgefeiltes Hilfesystem für alle Formularfelder auszutüfteln. (Bei nicht-trivialen Formularen empfiehlt sich trotzdem ein kontext-sensitives Hilfesystem.)

Wir haben bei vielen Programmen festgestellt, dass es hilfreich ist, dem Benutzer eine Demoversion zur Verfügung zu stellen, in der er experimentieren kann oder durch die er Schritt für Schritt durchgeleitet wird.

Auch Konsistenz hilft, ein Programm selbsterklärend zu machen. Wenn eine Warnung stets in der gleichen Darstellung an derselben Stelle erscheint, wird der Benutzer diese leichter als Systemmeldung erkennen.

4.3.2 Einfache Tauglichkeitsprüfung (Discount Usability Engineering)

In der Praxis werden die empfohlenen Tauglichkeitsprüfverfahren für Softwareentwicklungsprojekte nur selten verwendet. Dies liegt zum großen Teil an den Kosten, die mit den traditionellen Tauglichkeitsprüfverfahren verbunden sind. In der angesehenen Zeitschrift »Communications of the ACM« schätzten die Autoren Mantei und Teorey[5] 1988, dass sich die »Kosten, die erforderlich sind, um das Element menschlicher Faktor zur Softwareentwicklung hinzuzufügen« auf über 120.000 Dollar belaufen, ungeachtet der Größe des Softwareentwicklungsprojektes. Diese Schätzung bezog sich hauptsächlich auf die Bewertung fester Kosten, wie Testen und Auswertung, Praxistests, Produktanalysen usw., die sich nicht mit der Größe des Projekts und des Programmcodes während der Entwicklung ändern. Diese Zahl überschreitet

den Betrag für das gesamte Budget für Entwicklungsprojekte vieler Websites. Es ist nicht überraschend, dass ein Projektmanager beim Lesen solcher Summen Tauglichkeitsprüfverfahren ganz und gar ablehnt.

Jakob Nielsen’s Ansatz einer »einfachen Tauglichkeitsprüfung« (www.useit.com/papers/guerrilla_hci.html) zielt darauf ab, Tauglichkeitsprüfungen einfacher, billiger und weniger zeitaufwendig zu gestalten. Als er 1989 seine Methode untersuchte, fand er heraus, dass es für die meisten Projekte unnötig ist, alle herkömmlichen Tauglichkeitsprüfungen anzuwenden. Tatsächlich konnte er die Tauglichkeit bereits mit einer Reihe von Basisverfahren deutlich verbessern.

Nach dem Prinzip der frühen Fokussierung auf den Benutzer setzte er einige Techniken ein, die wir in den folgenden Abschnitten besprechen:

- Szenarien
- Vereinfachtes lautes Denken
- Heuristische Auswertung

Szenarien

Ein Szenario ist eine deutlich reduzierte Reihe von Funktionen oder Funktionalitäten, die einem Tauglichkeitstest unterzogen werden. Traditionell werden für Tauglichkeitsprüfungen komplexere Testfälle verwendet, die schwierig einzurichten und zu testen sind. Da unser Szenario klein ist, können Sie es ohne weiteres häufig ändern und es zum Testen verschiedener Versionen eines Prototyps verwenden.

Dieses Szenario wurde auf der Basis einer Aufgabenanalyse von realen Benutzern und ihrer Arbeit erstellt, womit es in Hinsicht auf die tatsächliche Verwendung des Systems so wirklichkeitsgetreu ist wie möglich. Typische Szenarien sind Aufgaben, wie »Dokument ausdrucken« oder »einen Flug nach Honolulu für den 1. Januar 2001 buchen«. Diese können vom Benutzer mithilfe einer vereinfachten Version des lauten Denkens getestet werden. Dieses Verfahren beschreiben wir im nächsten Abschnitt.

Vereinfachtes lautes Denken

In einer Studie über das laute Denken wird ein Benutzer überwacht, während er eine zuvor definierte Aufgabe in einem Szenario durchführt. Traditionell wurde diese Studien von Psychologen oder Tauglichkeitsexperten durchgeführt, welche die Tester auf Video aufnahmen und anschließend die Aufzeichnungen ausführlich analysierten. Auch diese anspruchsvolle Methode ist kostenintensiv und wirkt möglicherweise auf einige Entwickler einschüchternd. Es ist jedoch auch möglich, einen Test ohne die anspruchsvolle Testumgebung

durchzuführen. Dazu brauchen Sie nur einige reale Benutzer, denen Sie typische Testaufgaben geben und die Sie bitten, während der Durchführung der Testaufgabe laut zu denken.

Bei den vereinfachten Tests brauchen Sie idealer weise drei bis fünf Benutzer, während herkömmliche Verfahren 10 oder mehr Benutzer erfordern, um zuverlässige Ergebnisse zu erhalten. Das Hauptaugenmerk der einfachen Tauglichkeitsprüfung besteht jedoch nicht im Sammeln von statistisch gültigen Daten, sondern ist darauf ausgerichtet, Probleme in Bezug auf die Tauglichkeit so schnell wie möglich zu finden. Tatsächlich zeigen Studien, dass die meisten Tauglichkeitsprobleme innerhalb der ersten Tests aufgespürt werden können.

Anstatt die Testperson per Video aufzuzeichnen, halten die Versuchsbeobachter das Verhalten der Benutzer auf Papier fest. Das Aufzeichnen, Ansehen und Analysieren von Videobändern ist teuer und erfordert mindestens eine zusätzliche Person, welche die Kamera bedient. Die Analyse der Papiernotizen geht schnell vonstatten und ist trotzdem effizient. Die gesparte Zeit lässt sich besser zum Durchführen weiterer Tests und zum Testen verschiedener Iterationen einer Schnittstelle verwenden.

Heuristische Auswertung

Die heuristische Auswertung ist ein Verfahren, um Eignungsprobleme in einer Schnittstelle zu finden. Eine kleine Anzahl von Bewertern testet die Schnittstelle und bewertet ihre Übereinstimmung mit den Designprinzipien etablierter Benutzerschnittstellen (der Heuristik). Diese Prinzipien können die hier beschriebenen zehn empfohlenen heuristischen Methoden von Nielsen sein, die sich auf den Entwurf einer Benutzerschnittstelle beziehen, oder eine Reihe von benutzerspezifischen Prinzipien, die auf eine bestimmte Umgebung zugeschnitten sind.

Die Bewerter müssen keine Tauglichkeitsexperten sein. Selbst Laien können viele Tauglichkeitsmängel durch heuristische Auswertung aufdecken, und viele der verbleibenden Probleme lassen sich mit der vereinfachten Version des lauten Denkens entdecken. Da verschiedene Leute unterschiedliche Tauglichkeitsprobleme finden werden, kann es außerdem hilfreich sein, drei bis fünf Leute eine heuristische Auswertung vornehmen zu lassen. Ein Projektleiter sollte die Ergebnisse von den einzelnen Bewertern sammeln und einen ausführlichen Testbericht erstellen, der dann auf einem fachübergreifenden Treffen vorgestellt werden sollte.

Bei der heuristischen Methode wird nur die Schnittstelle auf Probleme überprüft. Die Bewerter verwenden nicht das eigentliche System. Sie nutzen die Heuristik als Prüfliste und versuchen, so viele Angriffe auf diese zu finden wie möglich. Dieser Ansatz sollte daher nur als Zusatz zu den anderen Tauglichkeitsprüfverfahren eingesetzt werden.

Zehn heuristische Tauglichkeitsprüfungen nach Jakob Nielsen

(`www.useit.com/papers/heuristic/heuristic_list.html`)

Sichtbarkeit des Systemstatus

Das System soll den Benutzer über entsprechende Rückmeldungen innerhalb einer angemessenen Zeit stets darüber informieren, was vor sich geht.

Abstimmung zwischen dem System und der realen Welt

Das System sollte keine systembezogenen Begriffe verwenden, sondern in der Sprache des Benutzers »sprechen« und dabei Wörter, Phrasen und Konzepte verwenden, die der Benutzer kennt. Es sollte praxisübliche Konventionen verwenden und Informationen in natürlicher und logischer Reihenfolge anzeigen.

Benutzerkontrolle und Freiheit

Benutzer wählen häufig versehentlich falsche Systemfunktionen und brauchen deutlich markierte »Notausgänge«, um den ungewünschten Zustand zu verlassen, ohne dabei einen langen Dialog durchlaufen zu müssen. Unterstützen Sie die Funktionen »rückgängig« und »wiederherstellen«.

Konsistenz und Standards

Benutzer sollten nicht raten müssen, ob verschiedene Wörter, Situationen oder Aktionen dasselbe bedeuten. Halten Sie sich an Plattformkonventionen.

Fehlervermeidung

Besser als gute Fehlermeldungen ist ein sorgfältiges Konzept, das in erster Linie verhindert, dass ein Problem auftritt.

Erkennen statt Erinnern

Gestalten Sie Objekte, Aktionen und Optionen deutlich erkennbar. Der Benutzer sollte sich nicht Daten von einem Teil eines Dialogs bis zum nächsten merken müssen. Anweisungen für die Bedienung eines Systems sollten sichtbar oder zumindest leicht aufzurufen sein.

Flexible und effiziente Verwendung

Beschleuniger, die der Laie nicht sieht, können die Interaktion für den erfahrenen Benutzer häufig beschleunigen, so dass das System erfahrene und unerfahrene Benutzer bedienen kann. Ermöglichen Sie dem Benutzer, häufig verwendete Aktionen auf seine Bedürfnisse anzupassen.

Ästhetisches und minimalistisches Design

Dialoge sollten keine Informationen enthalten, die irrelevant sind oder selten gebraucht werden. Jede zusätzliche Informationseinheit in einem Dialog lenkt die Aufmerksamkeit von den relevanten Teilen ab.

Hilfe für Benutzer zum Erkennen und Beheben von Fehlern

Fehlermeldungen sollten als Klartext (nicht Programmcode) ausgegeben werden, das Problem genau benennen und eine konstruktive Lösung vorschlagen.

Hilfe und Dokumentation
Es ist zwar besser, wenn das System ohne Dokumentation genutzt werden kann, aber manchmal kann es notwendig sein, eine Hilfe und Dokumentation bereitzustellen. Diese Informationen sollten leicht zu finden sein, auf die Aufgabe des Benutzers ausgerichtet sein, die konkreten auszuführenden Schritte beschreiben und nicht zu lang sein.

4.3.3 Tauglichkeit: Tun Sie es einfach

Gemäß Nielsen lauten zwei der wichtigsten Sprüche der reduzierten Tauglichkeitsprüfung: »Alle Daten sind Daten und irgendetwas ist besser als gar nichts, was die Tauglichkeit betrifft«. Wir möchten Sie ermutigen, die einfache Tauglichkeitsprüfung einmal auszuprobieren und sie bei Ihrer Entwicklung so häufig und regelmäßig wie möglich anzuwenden. Sie kostet nicht viel und fördert die Kenntnisse des Benutzers.

4.4 Zusammenfassung

In diesem Kapitel haben Sie die Grundlagen der Sitzungsverwaltung kennen gelernt. Sie haben erfahren, wie Sie Daten von einer Seite zur nächsten weiterleiten und wie Sie Ihre Benutzer erkennen. Sie können Sitzungen mit PHP 3.0 und 4.0 erstellen, wenn Sie verschiedene Speicherverfahren verwenden und wissen, wann Sie welches unter welchen Umständen verwenden müssen. Darüber hinaus haben wir Ihnen gezeigt, wie Sie die Daten sichern, die Ihnen Ihr Benutzer anvertraut hat, und wie Sie Ihre Website vor Missbrauch schützen. Sie kennen die Grundlagen der verschiedenen Verschlüsselungsmethoden und ihre Anwendungsgebiete.

Tauglichkeit ist für Sie jetzt kein Fremdwort mehr und wir hoffen, dass Ihnen jetzt bewusst ist, wie wichtig es ist, dass sich Ihre Website für ihre Aufgabe eignet. Sie kennen die meisten der üblichen Fehler und können Schwächen Ihres Schnittstellendesigns entdecken ohne dabei die Entwicklungskosten in die Höhe zu treiben.

Letztendlich sind es Ihre Benutzer, die über den Erfolg oder Misserfolg Ihrer Projekte entscheiden, und dieses Kapitel hat Ihnen gezeigt, wie Sie sie zufrieden stellen.

4.5 Literaturhinweise

[1] Siehe die historische Protokolldefinition unter www.w3.org/Protocols/HTTP/AsImplemented.html.

[2] Wiener, M.J. »Efficient DES Key Search.« Technical Report TR244, School of Computer Science, Carleton-Universität, Ottawa, Canada (May 1994).

[3] Fagan, M. E. »Design and Code Inspections to Reduce Errors in Program Development,« IBM Systems Journal, Bd. 15, Nr. 3 (1976): 182-211.

[4] Nielsen, Jakob. »Alertbox« (22. August 1999). Siehe www.useit.com/alertbox/990822.html.

[5] Mantei, M. M., und Teorey, T.J. »Cost/benefit analysis for incorporating human factors in the software lifecycle,« Communications of the ACM 31, 4 (April 1988): 428-439.

5 Grundlegende Webanwendungsstrategien

Wenn Menschen etwas als hübsch betrachten,
werden andere Dinge hässlich.
Wenn Menschen etwas als gut betrachten,
werden andere Dinge schlecht.

Kapitel 4, »Webanwendungskonzepte«, beschrieb die grundlegenden Unterschiede zwischen Webanwendungen und eigenständigen Skripten. Es zeigte Verfahren, mit denen sich das grundlegendste Problem beheben lässt: die Sitzungsverwaltung. Dieses Kapitel erläutert Strategien, um andere übliche Probleme, die beim Arbeiten an größeren Projekten auftreten, in den Griff zu bekommen. Wir stellen Ihnen einige Lösungen vor, die unserer Meinung nach zeitsparend, effektiv und einfach zu installieren sind. Trotzdem sollten Sie prüfen, ob sie wirklich Ihren Anforderungen genügen Dieses Kapitel nennt sich daher auch »Webanwendungsstrategien« und nicht etwa »Webanwendungslösungen« für einen Zweck. Es zahlt sich immer aus, mehr Zeit auf die Auswertung zu verwenden, als ein Projekt neu organisieren zu müssen. Wie in Kapitel 1 »PHP-Konzepte« erwähnt, ist Zeit die wertvollste Resource, von der wir niemals genug haben.

Am Anfang war das HTML-Formular. Fast jede Webanwendung erfragt Informationen vom Benutzer. Deshalb sollte ein besonderes Augenmerk auf Überprüfungsroutinen fallen. Sie brauchen eine generische Bearbeitungsroutine, wenn Sie das Rad nicht immer wieder neu erfinden möchten. Der erste Teil dieses Kapitels beschreibt das PHP-Standardformular und zeigt, wie Sie die Klasse `EasyTemplate` einsetzen, um Code und Layout voneinander zu trennen.

Über Schablonen wird die Zusammenarbeit in fachübergreifenden Entwicklungsteams verbessert. Dies ist jedoch nur ein kleiner Aspekt bei der Teamarbeit. Wir zeigen Ihnen deshalb auch, wie Sie Ihre Projekte organisieren und wie Sie Programme zur Versionskontrolle nutzen.

Auf dem Weg zum nächsten Kapitel machen wir dann einen kurzen Halt in der Marketingabteilung und erörtern die tatsächlichen Vorteile eines mehrstufigen Programms.

5.1 Die PHP-Normal Form

Wie überprüfen Sie Ihre Formulardaten? Mit JavaScript? Einem zweiten Skript zur Aktionsverarbeitung? Möglicherweise gar nicht oder nur teilweise?

Wie in Kapitel 4 erläutert, sollten Daten, die vom Benutzer in einem Formular oder einer Anfrage geschickt wurden, solange als »kontaminiert« betrachtet werden, bis sie von Ihrem Programm überprüft wurden. Deshalb sollten Sie die Eingaben besser überprüfen. Aber welches Verfahren eignet sich hierfür?

Eine übliche Methode ist JavaScript. JavaScript sollte jedoch nie die einzige Überprüfungsmethode sein, die Sie verwenden. Der Benutzer hat es vielleicht deaktiviert, um das Sicherheitsrisiko zu vermeiden, das mit der Skripterstellung auf der Clientseite verbunden ist, oder der Browser unterstützt kein JavaScript. Im schlimmsten Fall müssen Sie damit rechnen, dass Ihre Website-Benutzer die JavaScript-Funktionen deaktiviert haben. Da es auf den Browsern auch unterschiedliche Versionen von JavaScript geben kann, sind die komplizierteren Überprüfungsmethoden, z.B. die Überprüfung der Musterübereinstimmung mit regulären Ausdrucken, entweder gar nicht zu realisieren oder ein Alptraum für den Programmierer. Obwohl Sicherheit durch Verschleierung niemals ein gutes Prinzip ist, möchten Sie manchmal vielleicht nicht, dass der Benutzer mithilfe der View-Source-Funktion Ihre Überprüfungsregeln in seinem Browser sehen kann.

Wenn es sehr zuverlässig zu verwenden wäre, wäre JavaScript (oder allgemein eine Überprüfung auf der Clientseite) natürlich unsere erste Wahl. Der Hauptvorteil der Überprüfung auf der Clientseite besteht in der Geschwindigkeit. Die syntaktische Überprüfung könnte sofort erfolgen, so dass die Formulardaten nicht mehr an den Webserver geschickt, dort analysiert und anschließend als komplette Seite wieder an den Browser zurück geschickt werden müssten – alles eine sehr ermüdende und langsame Prozedur. Bisher kann JavaScript jedoch nur als zusätzliche Methode für die Überprüfung auf der Serverseite eingesetzt werden.

Damit kommt PHP ins Spiel. Selbst mit PHP gibt es viele Möglichkeiten der Formularüberprüfung. Die meisten neuen Benutzer werden sich für die direkte Methode mit zwei separaten Dateien entscheiden: eine mit dem HTML-Code des Formulars und eine mit dem PHP-Aktionsverarbeitungsskript. Dieses Verfahren ist traditionell für die Skripterstellung mit Perl/CGI vorgesehen. In den meisten Fällen lehnen wir dieses Szenario ab, weil es mehrere Nachteile hat:

- Sie brauchen zwei separate Dateien. So nichtig dies auch klingen mag, es kann zum Problem werden, wenn Sie größere Projekte verwalten, die Hunderte von PHP-Dateien enthalten.

- Es führt zu solchen Meldungen. wie »Ihr Formular hat einen Fehler, bitte klicken Sie auf die Schaltfläche »Zurück«, um zum Formular zurückzukehren«, die wir bereits in Kapitel 4 abgelehnt haben.
- Es ist einfacher, einige Formularfelder im Voraus auszufüllen, wenn sich Überprüfungsmethode und Formular in einer Datei befinden. Es ist unerheblich, woher die vorweg eingetragenen Daten kommen, vom Benutzer im Falle eines Fehlers oder von einer Datenbank für eine zu bearbeitende Seite.

Nach unserer Erfahrung ist die PHP-Normal Form, das Standardformular von PHP (das in Abbildung 5.1 zu sehen ist) die vielseitigste Alternative zur Bearbeitung von Formularen. Möglicherweise sind andere Strategien für Ihre Bedürfnisse besser geeignet, aber die Logik, die wir im Standardformular vorstellen, ist so allgemein, dass sie sich auch in den meisten anderen Formularen einsetzen lässt. Grundsätzlich vereint die PHP-Normal Form die Benutzerschnittstelle (das Formular) und die Programmlogik auf einer Seite, wobei HTML-Layout und Code trotzdem getrennt bleiben. Bei der ersten Anfrage nach der Seite wird das Formular angezeigt. Sobald der Benutzer das Formular weiterleitet, überprüft das Programm die Daten.

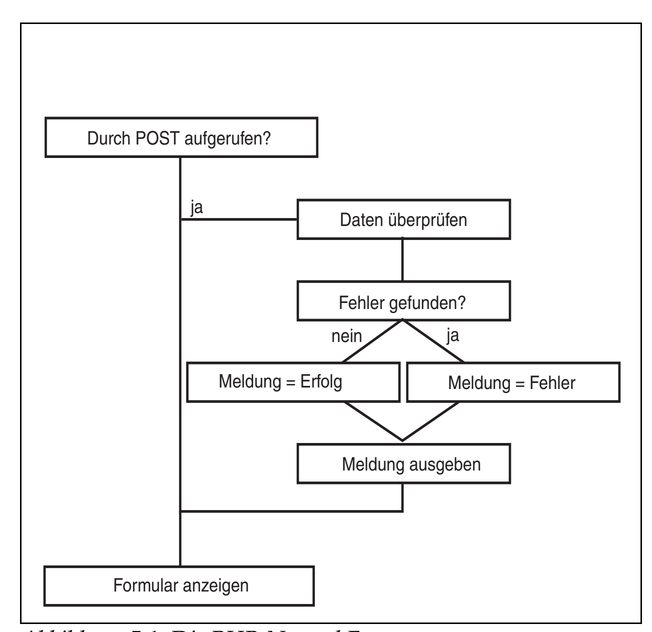

Abbildung 5.1: Die PHP-Normal Form

Lassen Sie uns nun beispielhaft ein einfaches Formular erstellen (Listing 5.1 zeigt den Programmcode). Hierin soll der Benutzer seinen Namen und seine Email-Adresse eintragen. Zu Beginn des Skripts prüft die Programmlogik, ob

das Skript beim Weiterleiten des Formulars aufgerufen wurde. Wenn dies der Fall ist, wird die CGI-Umgebungsvariable `$REQUEST_METHOD` auf `POST` gesetzt. Damit weiß das Programm, dass es mit der Überprüfung der Formulardaten beginnen soll.

```
/*
 *      mixed sprint_error(string string)
 *      Return a formatted error message or nothing if the passed argument
        ➥is empty
 */
function sprint_error($string)
{
    if(!empty($string))
    {
        return("<br><font color=\"red\"><b> ! </b></font>$string\
        ➥n");
    }
}
// Initialize $message
$message = "Please fill out this form:";

// Has the form been posted?
if($REQUEST_METHOD == "POST")
{
    // Initialize the errors array
    $errors = array();

    // Trim all submitted data
    while(list($key, $value) = each($form))
    {
        $form[$key] = trim($value);
    }

    // Check submitted name
    if(!is_clean_text($form["name"], 4, 30))
    {
        $errors["name"] = "The name you entered is not valid.";
    }

    // Check submitted email address
    if(!is_email($form["email"]))
    {
        $errors["email"] = "The email address you entered is not valid.";
    }

    print('<div align="center">');
```

```
    // Can the form be processed or are there any errors?
    if(count($errors) == 0)
    {
        print("The form has been processed.<p>");
        printf("Name: %s<br>", $form["name"]);
        printf("Email: %s<br>", $form["email"]);
        phpBook_footer("../");
        exit;
    }
    else
    {
        $message = "There were some errors. Please see below for details.";
    }

    print('</div>');
}

// Create a new EasyTemplate class
$template = new EasyTemplate("PHP_Normal_Form.inc.html");

// Assign template values
$template->assign("HEADER", $message);
$template->assign("ACTION", basename($PHP_SELF));
$template->assign("NAME_VALUE", isset($form["name"]) ? $form["name"] : "");
$template->assign("EMAIL_VALUE",  isset($form["email"]) ? $form["email"] :
➥"");
$template->assign("NAME_ERROR", sprint_error($errors["name"]));
$template->assign("EMAIL_ERROR",  sprint_error($errors["email"]));

// Print the template
$template->easy_print() or die($template->error);
```

Listing 5.1: Beispiel für ein PHP-Standardformular

> **Tipp:** `$REQUEST_METHOD` ist offensichtlich eine globale Variable und daher innerhalb von Funktionen nicht verfügbar. Innerhalb von Funktionen greifen Sie auf diese Variable zu, indem Sie das Array `$GLOBALS` in der Form `$GLOBALS["REQUEST_METHOD"]` verwenden oder über `global $REQUEST_METHOD` eine globale Anfrage starten.

Die weitergeleiteten Daten werden auf syntaktische Richtigkeit überprüft. Dazu werden zwei Funktionen aus der Datei `String_Validation.inc.php3` verwendet, die einige häufig verwendete Überprüfungsfunktionen für Zeichenketten bereitstellt (siehe Tabelle 5.1). Wird ein Fehler in den übertragenen Daten gefunden, dann wird im zugehörigen Feld `$errors` eine entsprechende

Fehlermeldung ausgegeben. Dies hilft später festzustellen, ob ein Fehler gefunden wurde. Wenn `count($errors)` größer ist als Null, reagiert das Programm entsprechend und verarbeitet das Formular nicht.

Funktion	Zweck
`is_alpha()`	Prüft, ob eine Zeichenfolge nur Buchstaben enthält; stellt optional die kürzeste und längste Zeichenfolge fest.
`is_numeric()`	Prüft, ob eine Zeichenfolge nur numerische Zeichen enthält; stellt optional die kürzeste und längste Zeichenfolge fest.
`is_alphanumeric()`	Prüft, ob eine Zeichenfolge nur alphanumerische Zeichen enthält; stellt optional die kürzeste und längste Zeichenfolge fest.
`is_clean_text()`	Prüft, ob eine Zeichenfolge einen erweiterten Satz von Zeichen enthält, der im Namen des westlichen Alphabets vorkommen kann (einschließlich aller Buchstaben, Umlaute und Anführungszeichen); stellt optional die kürzeste und längste Zeichenfolge fest.
`is_email()`	Prüft, ob eine Zeichenfolge eine syntaktisch gültige Email-Adresse ist (siehe später die Erörterung über die Grenzen dieser Funktion).
`contains_bad_words()`	Prüft, ob eine Zeichenfolge Wörter enthält, die im Array `$bad_words` innerhalb der Funktion definiert sind. Dies ist hilfreich für schwarze Bretter, Diskussionsräume usw., wo Sie Beleidigungen vermeiden wollen.
`contains_phone_number()`	Prüft, ob eine Zeichenfolge Telefonnummern enthält, d.h. Sequenzen von 10 und mehr Ziffern, die durch Klammern, Leerzeichen, Bindestriche oder Schrägstriche unterteilt sein können.

Tab. 5.1: Überprüfungsfunktionen für Zeichenketten aus String_Validation.inc.php3

Überprüfung einer Email-Adresse Die Überprüfung einer Email-Adresse ist ein übliches Problem. In unserem Ansatz überprüfen wir die syntaktische Richtigkeit mit folgendem regulären Ausdruck:

```
$ret = ereg(
          '^([a-z0-9_]|\\-|\\.)+'.
          '@'.
          '(([a-z0-9_]|\\-)+\\.)+'.
          '[a-z]{2,4}$',
          $string);
```

- Muss einen definierten Satz von Zeichen vor dem (@)-Zeichen enthalten.
- Ein @-Zeichen
- Ein Hostname mit einer Mindestlänge von zwei Zeichen
- Eine Domäne der obersten Ebene mit einer Mindestlänge von zwei Zeichen

In den meisten Fällen reicht diese Überprüfung aus. Aber es ist auf keinen Fall eine vollständige Überprüfung der offiziellen Syntax, die in RFC 822 definiert ist. RFC lässt Email-Adressen in der Form `"Name" mail@host.com` zu, die bei unserer Überprüfung abgelehnt würden. Unsere Überprüfung kontrolliert aber auch nicht, ob die Email-Adresse tatsächlich existiert.

Hin und wieder können Sie auch einen anderen Ansatz sehen, der zunächst stringenter erscheint, sich aber im realen Leben häufig als nicht nützlich erweist. Dies liegt daran, dass SMTP, das Simple Mail Transfer Protocol, eine Rückmeldung gibt, ob ein lokaler Benutzer existiert. Wenn wir einen SMTP-Kopf senden, der die Adresse des Empfängers oder den Benutzernamen auf dem lokalen System enthält, antwortet der SMTP-Server mit einem Statuscode von `250`, wenn der Benutzer existiert oder dem Statuscode `550`, wenn der Benutzer dem System unbekannt ist. So weit, so gut – theoretisch. Leider sind viele Mailserver so konfiguriert, dass sie alle Benutzer akzeptieren. Sie beantworten alle Anfragen mit »250 – Empfänger OK«. Wir haben es mit einem der Mailserver ausprobiert, die wir verwenden. Da unser Mailserver auf diese Weise konfiguriert war, erhielten wir die erwartete Antwort, dass alle Benutzer akzeptiert werden. Da SMTP ein Klartextprotokoll ist, können Sie dieses Verhalten sehr einfach überprüfen, indem Sie per Telnet eine SMTP-Sitzung simulieren:

```
bash-2.01$ telnet smtp.dnet.it 25
Trying 194.242.192.2...
Connected to ns.dnet.it.
Escape character is '^]'.
220 ns.dnet.it ESMTP Sendmail 8.9.3/8.9.3; Tue, 26 Oct 1999 14:02:43 +0200
(MET)
HELO www.profi.it
250 ns.dnet.it Hello www.profi.it [194.242.192.194], pleased to meet you
MAIL FROM: tobias@dnet.it
250 tobias@dnet.it... Sender ok
RCPT TO: this.user.doesnt.exist.for.sure
250 this.user.doesnt.exist.for.sure... Recipient ok
QUIT
221 ns.dnet.it closing connection
Connection closed by foreign host.
```

Eine bessere Idee wäre zu überprüfen, ob der MX(Mail eXchange)-Host existiert, indem Sie beispielsweise die Funktion `getmxrr()` benutzen. Dies kann die Überprüfung jedoch erheblich verlangsamen, wenn der DNS-Server des ent-

fernten Host einen langsameren Zugriff hat oder Lookups sich nicht im lokalen Cache befinden. Verwenden Sie diese Methode also nur mit äußerster Sorgfalt. Bedenken Sie, dass es mehr als einen MX-Host geben kann und es ausreicht, wenn einer von ihnen erreichbar ist.

Manchmal wird noch eine weitere Methode verwendet. Das PHP-Skript könnte einen Fingerzeig auf den Benutzer des entfernten Host setzen, um eine Email-Adresse zu überprüfen. Dies funktioniert natürlich nur, wenn (a) ein Fingerzeigserver eingerichtet wurde und (b) der Benutzername der E-Mail-Adresse entspricht. Dies geschieht nicht sehr häufig.

Abbildung 5.2 zeigt die Reaktion des Programms bei einer falschen Eingabe. Direkt unter dem fehlerhaften Eingabefeld wird eine Fehlermeldung angezeigt, wodurch der Benutzer eine sofortige Rückmeldung erhält. In allen Formularfeldern sind die Eingaben des Benutzers vorab eingetragen, so dass Daten nie verloren gehen. Wenn während der Überprüfungsphase kein Fehler entdeckt wurde, wird `count($errors)` auf Null gesetzt, und das Programm kann die Daten verarbeiten, d.h. sie in eine Datenbank speichern, sie per Email weiterschicken usw.

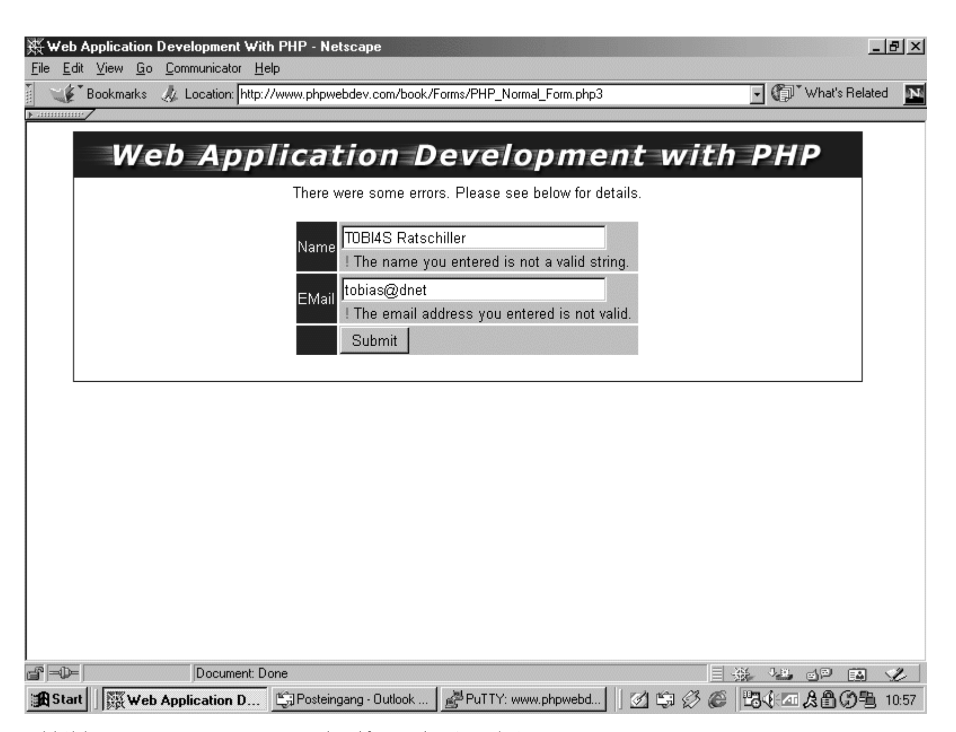

Abbildung 5.2: Das PHP-Standardformular in Aktion

Ein weiterer kleiner Trick, den wir für sehr nützlich halten, ist, alle Formulardaten in einem Array zusammenzufassen. Im Beispiel nennen wir es `$form`. Wenn Sie eine Überprüfungsregel auf jedes einzelne Element anwenden möchten, das vom Benutzer eingegeben wurde, wird sich dies als äußerst vorteilhaft erweisen: Sie müssen lediglich eine Schleife durch das Array laufen lassen. Im Beispiel verwenden wir dieses Verfahren, um mit `trim()` führende und angehängte Leerzeichen aus den Formulardaten zu entfernen.

Tipp: Im Gegensatz zu PHP 3.0 unterstützt PHP 4.0 mehrdimensionale Arrays in Formularen. Damit können Sie tatsächlich alle Formularvariablen, einschließlich `select multiple`-Felder in einem Array verwenden.

5.1.1 Verwendung von HTML-Schablonen

Möglicherweise haben Sie bemerkt, dass das Formularbeispiel keinen HTML-Code enthält. Genaugenommen haben wir HTML in eine separate Datei namens `PHP_Normal_Form.inc.html` ausgelagert. Hierbei handelt es sich um eine Vorlagendatei, welche die gesamte HMTL-Struktur und einige Schablonen-Tags, wie etwa `{HEADER}` enthält.

Vielleicht kommt Ihnen dies bekannt vor, weil Sie bereits mit der Klasse `FastTemplate` gearbeitet haben. `EasyTemplate` lässt die fortgeschrittenen Funktionen von `FastTemplate` weg; es stellt lediglich eine schnelle Schnittstelle zur Verfügung, um skalare Tags in Schablonen durch einen Zeichenkettenwert zu ersetzen. Da es sich auf diese einfache Aufgabe beschränkt, bietet es eine Leistungssteigerung von mehr als 120% im Vergleich zu `FastTemplate` für die syntaktische Analyse unserer Formularschablone.

Die von uns entwickelte Schablonenklasse hat nur drei Funktionen: `assign()`, `easy_parse()` und `easy_print()`, die in der folgenden Tabelle beschrieben werden:

Funktion	Beschreibung
`void EasyTemplate (string Schablone_Dateiname)`	Der Konstruktor der Klasse gibt im Argument den Dateinamen der Schablone zurück, die Sie verwenden möchten. Wenn ein Fehler auftritt, wird die Fehlermeldung in der Variablen `$error` ausgegeben.
`bool assign (String-Tag, Stringwert)`	Weist einem Schablonen-Tag einen Wert zu. Gibt bei erfolgreicher Zuweisung `true` zurück, bei einem Fehler `false`.

Funktion	Beschreibung
`mixed easy_parse()`	Analysiert die Schablone und gibt sie als Zeichenkette aus. Wenn ein Fehler auftritt, gibt die Funktion `false` zurück und setzt die Variable `$error` der Klasse auf eine sinnvolle Meldung.
`mixed easy_print()`	Analysiert die Schablone und druckt sie aus. Wenn ein Fehler auftritt, gibt die Funktion `false` zurück und setzt die Variable `$error` der Klasse auf eine sinnvolle Meldung.

Tipp: Ein Konstruktor gibt in PHP stets `void` zurück, egal was Sie mit `return()` zu tun versuchen.

Worin besteht der Vorteil Schablonen zu verwenden? Hauptsächlich in der Trennung von Darstellung und Code. Jeder Programmdesigner kann die Schablonendatei in seinem bevorzugten HTML-Autorenprogramm öffnen, ohne sich um den Skriptcode Sorgen machen zu müssen. Der Designer muss sich sogar überhaupt nicht mit HTML befassen.

Die Alternative wäre, den PHP-Code direkt in HTML einzubetten. Wenn Sie Ihr Programm umgestalten müssten, hätten Sie dann aber schwer zu kämpfen, um den HTML-Code manuell zu ändern.

Warum haben wir nicht `FastTemplate` verwendet? Zunächst einmal wäre es für diese Situation leicht übertrieben, und wenn eine solche Übertreibung einen Leistungsverlust von 120% bedeutet, überlegen wir nicht lange, ob wir nicht besser doch eine spezifischere Lösung finden.

Die Tatsache, dass wir das PHP-Standardformular zur Überprüfung von Formulardaten verwenden, bedeutet nicht, dass es die beste oder gar die einzige Methode ist, um Ihr spezifisches Problem zu lösen. Vielleicht wenden Sie ein, dass das PHP-Standardformular, wie wir es hier vorstellten, keine High-Level-Schnittstelle für Formulare bietet. Für jedes Formular, das Sie schreiben, müssen Sie ein Skript erstellen, um die Daten zu überprüfen. Sie sind ein kluger Kopf und haben sicherlich recht. Andere kamen bereits auf denselben Gedanken. Die Standard-PHP-Bibliothek `PHPLib` verfolgt einen objektorientierten Ansatz für die Handhabung von Formularen: `OOH Forms` und `tpl_form`.

Diese Klassen bieten eine objektorientierte Schnittstelle zu HTML-Formularen. Sie können Benutzereingaben auf der Clientseite mit JavaScript überprüfen und auf der Serverseite mit regulären PHP-Ausdrücken. Mit `OOH Forms` müssen Sie nicht direkt in HTML-Code schreiben, sondern können mithilfe seiner Funktionen Formularelemente erstellen. Dieses objektorientierte Verfahren kommt einigen Leuten entgegen, und wir möchten Sie ermuntern, sich dieses Paket einmal anzusehen. Die API ist zwar recht komplex, aber sehr gut

auf der Website dokumentiert, und zwar unter `http://phplib.netuse.de`. Wir fürchten, dass die Automatisierung der Formularüberprüfung im realen Leben nicht funktioniert, weil die Situationen häufig sehr komplex sind. Einige Formulareingaben hängen von zuvor eingetragenen Daten ab, verwenden komplexe Zeichenketten-Überprüfungsregeln, die sich nicht mit regulären Ausdrücken ausdrücken lassen, oder beziehen auch andere externe Quellen wie Datenbanken ein. Dies lässt sich nicht hinreichend abstrahieren, um vollständig generische Überprüfungsroutinen zu erstellen.

Hinweis: Zum Zeitpunkt der Erstellung dieses Buches gab es bereits erste Bemühungen, eine Schablonen-API in PHP 4.0 zu integrieren. Dies ist sicherlich zu begrüßen, da es eine standardisierte und offene Handhabung von Schablonen ermöglicht.

5.2 Projektstruktur

Die Zeit kommt, da ein Zwölf-Stunden-Tag, Wochenendarbeit sowie Tonnen von Kaffee und ebensoviel Cola nicht ausreichen, dass ein einzelner Entwickler ein Projekt rechtzeitig fertig stellt. Für den Durchschnittsprogrammierer ist dies ein bedeutender Einschnitt in seinem Leben – kein einsames Einhacken seines eigenen Codes mehr und kein cooler Programmierstil mehr, den nur er lesen kann, nicht mehr sein eigener Boss sein, der ab und zu aus dem Untergrund auftaucht, um sich eine Pizza abzuholen.

Nein – so nicht. Nun muss er sich Gedanken machen: über Standards, Stil und Projektverwaltung. Als Dankeschön kann er vielleicht zu den normalen Arbeitsstunden zurückkehren und sich einige Zeit freischaufeln, die er für nicht-arbeitsbezogene Dinge verwendet. Wer weiß, vielleicht kann er sogar am sozialen Leben wieder teilnehmen?

5.2.1 Zusammenarbeit im Team

Mit dem Internet entstand die Möglichkeit, über weite Entfernungen hinweg zu kommunizieren und zusammenzuarbeiten, egal wo sich der geographische Standort der einzelnen Teammitglieder tatsächlich befand. Dies eröffnet verteilten Teams neue Chancen. Große Firmen sind heutzutage nicht mehr durch geographische Grenzen eingeschränkt. Tatsächlich entwickeln viele Unternehmen Softwareanwendungen, indem sie Expertenwissen aus vielen verschiedenen geographischen Regionen miteinander kombinieren.

Für die effektive Verwaltung der verteilten Softwareentwicklung ist eine gute Projektleitung von immenser Bedeutung. Wenn ein neues Projekt gestartet wird, wissen die Leute zunächst einmal nichts. Sie wissen nicht, was zu tun ist

und wann es zu tun ist. Sie brauchen jemanden, der ihre Aktivitäten koordiniert, Verantwortlichkeiten zuweist und den Fortschritt und die Ergebnisse überwacht. Sie brauchen einen Projektleiter.

Die Projektverwaltung für die Entwicklung eines Programms ist ein so umfassendes Gebiet, dass wir es in diesem Buch leider nicht behandeln können. Es gibt viele gute Methoden und Mittel, und wir können Ihnen nur empfehlen, diese anzusehen, wenn Sie auf diesem Gebiet noch keine Erfahrung haben. Sie reichen von Werken wie »The Mythical Man Month« bis zu Methoden wie dem »Personal Software Process«.

Wir werden uns auf die einzelnen Stufen eines Projektes konzentrieren, die eine technische Entwicklung durchläuft. Sie haben bereits in Kapitel 3, »Programmentwurf« gesehen, wie Sie eine Anwendungsprogrammierschnittstelle (API) konzipieren können. Genau genommen war dies bereits ein eigenes Miniprojekt. Wir haben dabei im Stillen einige Annahmen gemacht: Wie wird der Code in Verzeichnissen organisiert, wer ist im Entwicklungsteam, wie werden verschiedene Versionen der API gepflegt?

In realen (größeren) Projekten müssen Sie diese Entscheidungen selbst treffen. Da diese die Grundlage für Ihr Projekt bilden, sollten Sie sich sorgfältig vorbereiten, bevor Sie etwas beschließen, was sich später möglicherweise als unpassend herausstellt.

5.2.2 Verzeichnisstruktur

Die grundlegendste und trotzdem häufig vernachlässigte technische Entscheidung, die Sie zu Beginn der Quellcodeentwicklung für ein neues Projekt treffen müssen, betrifft die zu verwendende Verzeichnisstruktur. In der Regel verwenden wir folgende Struktur:

```
/home/www/phpwebdev.com/live/cgi-bin
                             htdocs
                             htpasswd
                             include
                             logs
/home/www/phpwebdev.com/staging/cgi-bin
                                htdocs
                                htpasswd
                                include
                                logs
/home/www/phpwebdev.com/dev/cgi-bin
                            htdocs
                            htpasswd
                            include
                            logs
```

Wenn Sie auf die erste Verzeichniseinheit blicken, stellen Sie fest, dass es unterschiedliche Verzeichnisse für eingeschlossene Dateien (Bibliotheken, Konfigurationsdateien, Schablonen) und Passwortdateien (zu verwenden mit `.htaccess`-Dateien) gibt. Passwortdateien sollten niemals über das Web zugänglich sein, da die darin enthaltenen Passwörter nur mit schwachen Standardalgorithmen verschlüsselt sind und daher leicht von entsprechenden Programmen entschlüsselt werden könnten. Die enthaltenen Dateien müssen nicht unbedingt vor dem Zugriff von außen geschützt werden; es empfiehlt sich jedoch, sie außerhalb des Dokumentenstamms zu speichern. Selbst wenn der Webserver ein Dokument aufgrund einer falschen Konfiguration nicht analysieren kann, gibt es möglicherweise einige Bibliotheken, die wichtige Betriebsgeheimnisse, Systeminformationen oder innovative Algorithmen enthalten und nicht für alle Webbenutzer sichtbar sein sollen.

Tipp: Wir verwenden in allen Projekten eine Datei namens `configuration.inc.php3`, die Konfigurationsdaten für den Umfang eines Projekts definiert. Sie beschreibt außerdem den Basispfad des Projekts. Dies vereinfacht, später zusätzliche Dateien im Skript einzufügen, die sich in Unterverzeichnissen befinden.

Es gibt drei verschiedene Verzeichnisgruppen: `live`, `dev` und `staging`.

- Das Unterverzeichnis `live` enthält die Produktionsumgebung (die tatsächliche Website).
- `dev` wird für den Entwicklungsserver verwendet.
- `staging` bildet den Übergang von `dev` zu `live` und wird für die Qualitätssicherung und die Endabnahme vor der Freigabe verwendet.

Die Trennung von Entwicklungs- und Anwendungsserver ist auf größeren Websites unerlässlich. Sie können es sich einfach nicht leisten, Webanwendungen »live« zu bearbeiten. Ein Skriptfehler würde sich unmittelbar auf Hunderttausende von Benutzer auswirken. Bedenken Sie, was passieren könnte, wenn diese Skriptfehler zu Unvereinbarkeiten von Daten in einer Produktionsdatenbank führen würde.

Die Lösung besteht demnach in der Unterscheidung zwischen einem Entwicklungsserver und einem Produktionsserver. In unserem Beispiel befinden sich beide auf derselben physikalischen Maschine. Für den Entwicklungsserver könnten Sie einen Zugriff mit entsprechenden Zugriffsbeschränkungen unter `dev.phpwebdev.com` ermöglichen, z.B. mit IP-basierter Filterung. Auf größeren und kritischeren Systemen ist es in der Regel besser, den Entwicklungsserver auf einen zweiten, identisch konfigurierten Server zu verlagern. Damit können Sie Softwareaktualisierungen, Neukonfigurationen des Betriebssystems, Hardwareänderungen usw. leichter testen.

Doch wozu dient der Dritte im Bunde, der Übergangserver? Ist das nicht etwas übertrieben? Nicht, wenn Ihr Team groß genug ist, um gezielt Personal für die Qualitätssicherung (QA) oder die Sicherheitsüberprüfung abzustellen. Bei einer komplexen Website brauchen Sie häufig viel Zeit, um eine geänderte Version zu testen, bevor Sie diese freigeben. Sie können die Entwickler nicht dazu verdonnern, tatenlos herumzusitzen, während das Qualitätssicherungspersonal den Entwicklungsserver unter die Lupe nimmt. Ein weiterer Grund, warum Sie einen Übergangsserver in Ihre Konstellation integrieren sollten, ist, dass Entwickler sich sicherer fühlen und unbefangener Änderungen am Entwicklungsserver vornehmen. Sie brauchen sich nicht mehr große Sorgen zu machen, dass sie den Code eines anderen Entwicklers versehentlich zerstören. Wenn ein Problem auftritt, können Sie dieses lösen, bevor der Projektleiter die Entwicklungsversion an den Übergangserver weiterleitet. Wenn jemand den Übergangsserver zum Absturz bringt, muss er natürlich trotzdem die traditionellen dreißig Liegestützen machen.

Wie sieht nun ein typischer Entwicklungslebenszyklus in dieser Konstellation aus?

1. Das neue Projekt wird gestartet, indem eine Verzeichnisstruktur erstellt und ein System zur Versionsverwaltung errichtet wird (mehr zur Versionsverwaltung weiter unten).
2. Entwickler testen eine Kopie des Programmzweigs Entwicklung.
3. Die Entwickler leiten ihre Änderungen an den Programmzweig Entwicklung weiter.
4. Sobald ein wichtiger Meilenstein erreicht wurde und das System zum Testen bereit steht, transferiert der Projektleiter den Entwicklungscode zum Übergangsserver.
5. Das Qualitätssicherungs- und Sicherheitspersonal führen auf dem Übergangsserver Tests durch. Fehler und Probleme werden an die Entwickler weitergeleitet.
6. Die Entwickler beheben die Fehler und leiten das Projekt solange an die Qualitätssicherung zurück, bis keine Fehler mehr gefunden werden.
7. Für die Freigabe wird der Übergangsserver auf den Produktionsserver kopiert. Vergessen Sie nicht, die Freigabe gebührend zu feiern!

Bedenken Sie, dass wir hier nur über den Teil des Projektes reden, der sich mit der Entwicklung des Programmcodes befasst. Das gesamte Projekt durchläuft natürlich alle Zyklen der Softwareentwicklung:

1. Projektstart
2. Analyse
3. Entwurf

4. Technische Spezifikation und Entwurf der Datenbankstruktur
5. Implementierung (dies ist unser Thema)
6. Qualitätssicherung
7. Freigabe

5.3 Versionsverwaltung mit CVS

Wenn mehrere Entwickler am selben Projekt arbeiten, entstehen möglicherweise Versionskonflikte. Dies ist umso wahrscheinlicher, wenn die Entwickler nicht im selben Gebäude arbeiten, sondern über mehrere Gebiete und Ländergrenzen verteilt sind. Was passiert, wenn zwei Entwickler gleichzeitig dieselbe Datei bearbeiten? Die Änderungen des einen Entwicklers werden unweigerlich überschrieben. Was passiert, wenn das von John verfasste Skript nicht mehr funktioniert, nachdem es von Jane geändert wurde? John muss sich die Mühe machen herauszufinden, was Jane geändert hat.

An dieser Stelle machen sich Versionsverwaltungssysteme bezahlt. Die Software »erinnert sich« an die vorhergehenden Versionen einer Datei und ermöglicht Ihnen, auf eine ältere Version zurückzugreifen (eine Art von erweitertem Wiederherstellen). Außerdem benachrichtigt Sie das Versionskontrollprogramm, wenn andere Entwickler Dateien bearbeitet haben, an denen Sie gerade arbeiten, und gibt Ihnen die Möglichkeit, auf Konflikte zu reagieren. Schließlich merkt es sich, wer bestimmte Änderungen vorgenommen hat.

CVS ist ein Programm, das all diese Fähigkeiten besitzt – und sogar kostenlos ist (Open Source). Nicht nur die Entwicklung von PHP selbst wird durch CVS verwaltet, sondern auch erfolgreiche Projekte, wir bekannte Websserver (Apache), das Mozilla-Projekt und KDE vertrauen auf CVS, um die Arbeit von Dutzenden einzelner Entwickler zu koordinieren. Auch der XML-Quellcode dieses Buches wird mit CVS gepflegt. Die Autoren leben in Deutschland (Till) bzw. Italien (Tobias), der Verlag (New Riders) sitzt in den USA, der technische Redakteur lebt in Australien, und alle greifen auf einen zentralen Datenbestand namens phpBook auf einen CVS-Server zu, der sich in Norditalien befindet. Zu dem Datenbestand gehören alle Texte sowie einige PHP-Werkzeuge, die dazu dienen, einheitliche Versionen der einzelnen Kapiteldateien zu erstellen, einige Ressource-Dateien für XMetal (auf die wir in Kürze eingehen), die Codebeispiele und die Abbildungen. Auf unseren lokalen Systemen verwenden wir WinCVS oder ein Befehlzeilen-CVS, um die Dateien zu verwalten, während wir den Text mit XMetaL, SoftQuad's XML-Editor, auf recht bequeme Art und Weise bearbeiten können.

Viele Softwareprojekte haben ein ähnliches Szenario. Die Entwickler sind über die ganze Welt verteilt. CVS eignet sich hervorragend für internetbasierte Softwareentwicklung. Sie ist eine Client/Server-Anwendung, die das Netz als Transportschicht verwendet und den Quellcode zentral pflegt und verwaltet. Innerhalb des gesamten Datenbestandes werden die einzelnen Projekte in Modulen organisiert. Jeder Entwickler muss sich für das Modul, mit dem er aus diesem Datenbestand arbeiten möchte, anmelden, arbeitet dann aber mit der lokalen Version. Auf diese Weise werden die Onlinezeiten erheblich reduziert, insbesondere wenn Sie einen Internetzugang über Wählleitung verwenden müssen. Wenn der Entwickler alle Änderungen vorgenommen hat, kopiert er die aktualisierte Datei wieder in das zentrale Datenarchiv. Den Rest erledigt das CVS-Programm:

- Dieses weist jeder Änderung eine Versionsnummer zu,
- macht für jede geänderte Version einen Eintrag ins Protokoll,
- protokolliert, wer eine Datei zur Bearbeitung ausgelesen hat,
- und gibt eine Warnung aus, wenn andere eine Datei bearbeitet haben, die Sie abmelden möchten.

Lassen Sie uns eine typische CVS-Sitzung zu Beginn eines Projektes ansehen. Es arbeiten zwei Entwickler an der Implementierung einer API, John und Jane. Wir gehen davon aus, dass ihr Projektleiter einen vollständigen CVS-Server eingerichtet hat, ein zentrales Modul namens `f-api` erzeugt hat und die CVS-Umgebungsvariablen in ihren Shell-Accounts entsprechend gesetzt hat. (Weitere Informationen zur Installation auf einem CVS-Server oder Client finden Sie im CVS-Referenzhandbuch.)

Zunächst müssen sich beide Entwickler auf dem CVS-Server anmelden und eine Kopie des Moduls auf ihr lokales System laden. Dies geschieht mit dem Befehl `cvs checkout`, der ein Verzeichnis erzeugt, das nach dem Modul benannt ist (in unserem Beispiel `./f-api`):

```
john@dev:/mnt/daten/home/john > cvs login
(Logging in to john@www.phpwebdev.com)
CVS password: <password>
john@dev:/home/john > cvs checkout f-api
cvs server: Updating f-api
U f-api/config.inc.php3
U f-api/f-api_read.php3
U f-api/f-api_write.php3
```

Nachdem beide Entwickler jeweils eine lokale Kopie des gesamten Projekts haben, können sie die Dateien bearbeiten. Sie haben vereinbart, dass John an `f-api_read.php3` und Jane an `f-api_write.php3` arbeitet. Dies ist ein wichtiger Punkt: CVS kann die Absprache im Team nicht ersetzen.

Nach einem ersten Entwicklungsdurchgang sendet John die Datei wieder zum zentralen Datenarchiv zurück. Da niemand anderer die Datei in der Zwischenzeit bearbeitet hat, braucht er hierfür nur einen einzigen CVS-Befehl:

```
john@dev:/home/john/f-api > cvs commit f-api_read.php3
```

Daraufhin zeigt CVS den Standardtexteditor des Systems an – auf dem UNIX-System ist es häufig `vi`; auf Windows `Notepad` – und wartet auf einen Protokolleintrag, der dieser geänderten Version zugeordnet wird. Diese Meldungen können sich andere Entwickler ansehen. Deshalb sollte sie exakt beschreiben, welche Arbeiten durchgeführt wurden. Nachrichten, wie »Datei geändert« oder »neue Funktion«, sind nicht sehr hilfreich und sollten daher vermieden werden. Versuchen Sie, so präzise und klar wie möglich eine Zusammenfassung Ihrer Arbeit zu geben, und denken Sie dabei an die Richtlinien, die wir in Kapitel 1 »Entwicklungskonzepte« für eingebettete Quellcodekommentare aufgestellt haben. Einige Entwicklungsteams verwenden die Protokollnachrichten sogar, um den Kunden automatisch über die aktuellen Arbeiten zu informieren. En gutes Beispiel für eine aussagekräftige Protokollnachricht ist folgende:

```
Fixed bug #42; implemented additional checks on user-data (int, date),
see spec p25
```

Im Gegensatz zu anderen Versionskontrollsystemen, wie RCS oder Microsoft's Visual Source Safe, sperrt CVS die Dateien nicht, sobald Sie diese abgemeldet haben. Dies bedeutet, dass mehrere Entwickler dieselbe Datei gleichzeitig bearbeiten können, was in unseren Augen ein großer Vorteil ist. In unserem Szenario könnten sowohl John als auch Jane die Datei `config.inc.php3` gleichzeitig bearbeiten. Sobald John eine Datei wieder im System anmelden möchte, die von Jane bereits geändert und zurückgeschickt wurde, erhält er eine Warnung und muss seine lokale Kopie aktualisieren, bevor er die Datei zurück übertragen kann. In einfachen Fällen, z.B. wenn John eine Funktion zu Beginn und Jane eine Funktion am Ende der Datei bearbeitet haben, wird CVS die beiden Versionen automatisch vermischen und John kann die kombinierte Version sofort zurückschicken. In anderen Situationen ist eine automatische Vermengung vielleicht nicht möglich. Nehmen wir beispielsweise folgende Originalzeile aus einer Konfigurationsdatei:

```
define("FT_ZIP_ARCHIVE", "ZIP_ARC"); // GZip Archive
```

John ändert die Konstante und schickt die Datei zurück:

```
define("FT_ZIP", "ZIP_ARC"); // GZip Archive
```

Jane verbessert den Kommentar in derselben Zeile:

```
define("FT_ZIP_ARCHIVE", "ZIP_ARC"); // GZip Archive (not PkZip
compatible!)
```

Allerdings arbeitet sie jetzt nicht mehr an der aktuellen Version der Datei, so dass eine Vermengung offensichtlich notwendig ist. Wenn Jane versucht, Ihre Version zurück zu übertragen, gibt CVS folgende Warnung aus:

```
jane@dev:/home/jane/f-api > cvs commit config.inc.php3
cvs server: Up-to-date check failed for 'config.inc.php3'
cvs [server aborted]: correct above errors first!
cvs commit: saving log message in /tmp/cvs07789baa
```

Dies bedeutet, dass Jane ihre lokale Version aktualisieren muss, bevor sie diese an das zentrale Datenarchivr zurückschicken kann:

```
jane@dev:/home/jane/f-api > cvs update config.inc.php3
RCS file: /usr/local/cvsroot/config.inc.php3/config.inc.php3,v
retrieving revision 1.3
retrieving revision 1.5
Merging differences between 1.3 and 1.5 into config.inc.php3
rcsmerge: warning: conflicts during merge cvs
server: conflicts found in config.inc.php3
C config.inc.php3
```

Hier haben wir einen Fall, bei dem ein automatischer Abgleich nicht möglich ist. Statt dessen erzeugt CVS eine spezielle Version der Datei, die Markierungen enthält, die auf die widersprüchlichen Abschnitte hinweisen. In diesen markierten Abschnitten zeigt CVS die lokale Version des betreffenden Abschnitts und die Version aus dem zentralen Archiv:

```
<<<<<<< config.inc.php3
      define("FT_ZIP_ARCHIVE", "ZIP_ARC"); // GZip Archive (not PkZip
compatible!)
      =======
      define("FT_ZIP", "ZIP_ARC"); // GZip Archive
>>>>>>> 1.3
```

Die erste Gruppe zeigt Janes Version, die zweite die Version aus dem Archiv. Es ist nun Janes Aufgabe, den Code durchzusehen und die Konflikte zu beheben. Dies kann einfach darin bestehen, dass nur die Änderungen des anderen Entwicklers in der lokalen Version hinzugefügt werden müssen, oder aber auch so komplex, dass sie mit anderen Projektmitgliedern in einem Telefongespräch oder einer Besprechung geklärt werden müssen. Auch in unserem einfachen Beispiel müsste Jane wahrscheinlich mit John Rücksprache halten, um zu erfahren, warum er die Konstante von `FT_ZIP_ARCHIVE` auf `FT_ZIP` gekürzt hat.

Sobald Jane alle problematischen Stellen bereinigt hat, kann sie die Datei an das Archiv zurückschicken, und CVS wird es akzeptieren:

```
jane@dev:/home/jane/f-api > cvs commit config.inc.php3
```

Nun kann John wiederum seine lokale Version mithilfe der vereinheitlichten Version aus dem Archiv aktualisieren und damit diese Bearbeitungsstufe beenden.

Manuelle Abgleiche kommen nur selten vor, wir haben festgestellt, dass Sie diese meistens vermeiden können, wenn Sie eine entsprechende Kommunikation zwischen den Teammitgliedern pflegen und fördern. Die seltenen Konflikte, die tatsächlich Kopfschmerzen bereiten, sind nahezu ausschließlich auf mangelnde Kommunikation zwischen den Entwicklern zurückzuführen. Dieses Problem kann kein Versionskontrollsystem der Welt beheben.

Nachdem Jane es mit Konflikten zu tun hatte, ist sie vorsichtig geworden und möchte den Status der Datei `f-api_write.php3` überprüfen. Deshalb gibt sie den CVS-Befehl `status` aus:

```
jane@dev:/home/jane/f-api > cvs status f-api_write.php3
```

Durch diesen Befehl erfährt sie, dass ihre lokale Kopie die derzeit aktuelle ist. CVS hat folgende Statuscodes:

Code	Beschreibung
`Up-to-date`	Die lokale Kopie ist identisch mit der Kopie auf dem CVS-Server.
`Locally Modified`	Die lokale Kopie wurde geändert, aber noch nicht zum Archiv übertragen.
`Locally Added`	Diese Datei wurde nur im lokalen Verzeichnis hinzugefügt.
`Locally Removed`	Diese Datei wurde nur im lokalen Verzeichnis gelöscht.
`Needs Checkout or Needs Patch`	Die Version im zentralen Archiv ist neuer als die lokale Kopie, die aktualisiert werden müsste.
`Needs Merge`	Die Version im zentralen Archiv ist neuer als die lokale Kopie, die ebenfalls geändert wurde. Beide würden nach der Aktualisierung abgeglichen.
`File had conflicts on merge`	Es gab einen manuellen Abgleich, und die daraus resultierenden Konflikte wurden noch nicht gelöst.
`Unknown`	Diese Datei gehört nicht zur CVS-Versionskontrolle.

Jede geänderte Version, die von einem Entwickler an das zentrale Archiv geschickt wird, erhält automatisch eine Versionsnummer. Wie wir bereits erwähnt haben, können Sie dadurch jede Version einer Datei wieder herstel-

len. Mithilfe des Befehls `cvs diff` kann ein Entwickler sogar Unterschiede zwischen beliebigen Versionen ansehen, ohne die Datei auszulesen. Standardmäßig wird der Befehl `cvs diff` eingesetzt, um die lokal geänderte Version mit der Version im zentralen Archiv zu vergleichen:

```
jane@dev:/home/jane/f-api > cvs diff f-api_write.php3
```

Durch Verwendung der Option `-r`, die in den meisten CVS-Befehlen zur Versionsangabe eingesetzt werden kann, ist Jane jetzt in der Lage, die lokale Kopie mit der zentralen Version 1.1 (der ursprünglichen Version) zu vergleichen:

```
jane@dev:/home/jane/f-api > cvs diff -r 1.1 f-api_write.php3
Index: config.inc.php3
  ===================================================================
RCS file:
/usr/local/cvsroot/config.inc.php3/config.inc.php3,v
retrieving revision 1.1
retrieving revision 1.3
diff -r1.1 -r1.5 3c3
< define("FT_ZIP_ARCHIVE", "ZIP_ARC"); // GZip Archive
---
> define("FT_ZIP", "ZIP_ARC"); // GZip Archive (not PkZip compatible!)
```

Auch wenn sich CVS für den Abgleich verschiedener Versionen einer Datei meistens gut eignet, sollten Sie dies unter bestimmten Umständen besser vermeiden. Wenn der Befehl `cvs diff` beispielsweise umfangreiche Änderungen auf dem entfernten Server anzeigt, während auf dem lokalen System nur wenige Änderungen vorgenommen wurden, empfiehlt es sich möglicherweise, die lokale Version zugunsten der Änderung im zentralen Archiv zu verwerfen. Hier sollten Sie besser die lokale Datei löschen und die Datei vom Server erneut auslesen.

5.3.1 CVS-Optimierung: Grafische Benutzeroberflächen und CVSweb

Wenn Sie sich mit den grundlegenden CVS-Befehlen vertraut gemacht haben, die wir im vorangegangenen Abschnitt beschrieben haben, stehen Ihnen alle Möglichkeiten offen, die ein Kommandozeilenwerkzeug bietet: Sie können CVS auf entfernten Shells verwenden, Prozesse mit Shell-Skripten automatisieren und Ihren Kollegen zeigen, dass Sie Ihren Job wirklich beherrschen. Für die tägliche Arbeit bevorzugen wir jedoch eine grafische Benutzeroberfläche.

Für Windows-Rechner ist WinCVS, das Sie unter `www.wincvs.org` erhalten, ein sehr hilfreiches Programm, um die Arbeit zu erledigen (siehe Abbildung 5.3). Es verbirgt die Komplexität von CVS nicht, sondern gibt Ihnen einen bequemen Zugriff auf die am häufigsten benötigten Funktionen über eine grafische Benutzeroberfläche.

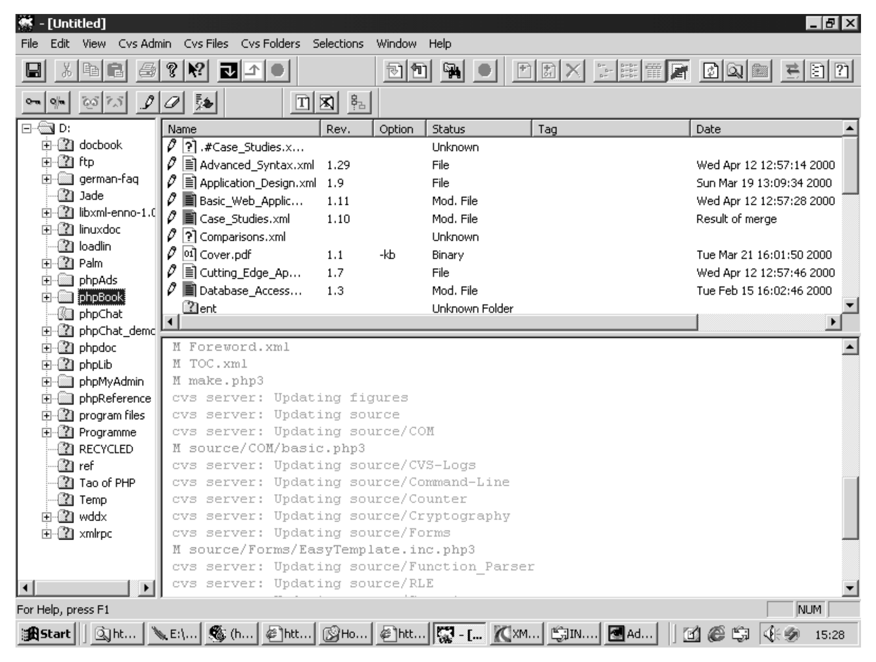

Abbildung 5.3: WinCVS

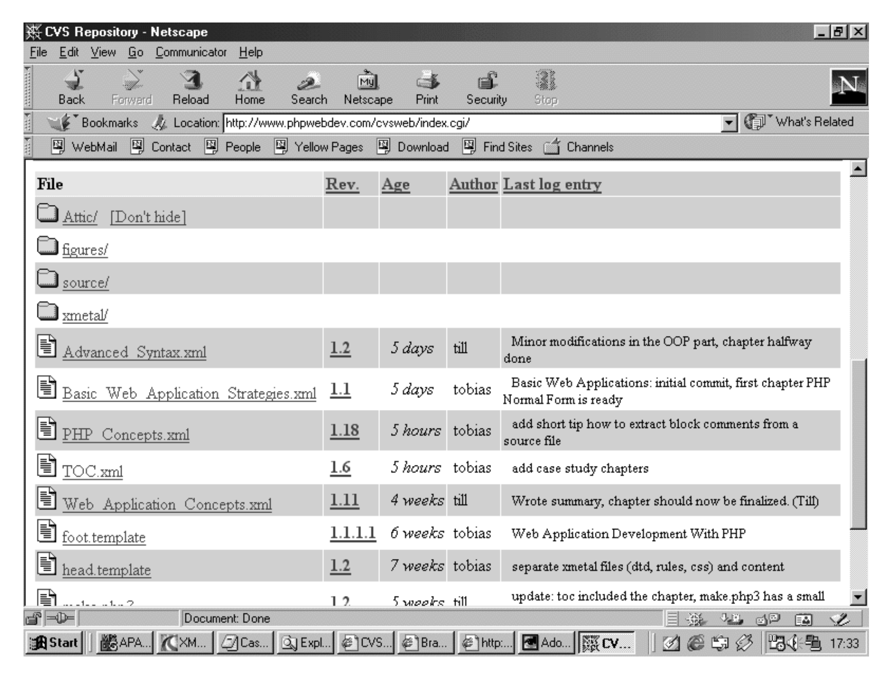

Abbildung 5.4: Die CVSweb-Schnittstelle

Ein weiteres nützliches Werkzeug ist CVSweb, ein in Perl geschriebenes CGI-Skript, das Ihnen einen Überblick über Ihre CVS-Archive gibt. Es vereinfacht die Verwaltung größerer Projekte, indem es jede Datei einer Liste zusammen mit der letzten Protokollnachricht, dem Änderungsdatum und dem Autor anzeigt (siehe Abbildung 5.4). Sie können alle geänderten Versionen einer Datei anfordern. Unterschiede zwischen beliebigen Versionen werden farbig dargestellt.

5.3.2 Erweitertes CVS

Natürlich gibt es einen Grund, dass das CVS-Referenzhandbuch von Per Cederqvist so ein großes Ungeheuer ist. Die grundlegenden Befehle sind selbst von Entwicklern schnell zu erlernen, die sich mit CVS gar nicht auskennen. Die erweiterten Funktionen von CVS bedürfen jedoch einer ausführlicheren Beschreibung.

Tags und Versionszweige

CVS verwendet beispielsweise das Konzept der Tags und Zweige. CVS-Tags sind recht einfach zu verstehen. In vorigen Beispiel haben John und Jane eifrig gearbeitet, bis sie die letzte Projektstufe erreicht und Version 1.0 ihrer F-API freigegeben haben. Zwei Tage später entdeckt ein wichtiger Kunde einen Fehler in der API. Es ist im Prinzip ein leicht zu behebender Fehler, aber die Entwickler haben die Entwicklung bereits fortgesetzt und können den aktuellen Quellcode nicht als stabile Version freigeben. Wenn sie den Fehler beheben wollten, müssten sie zunächst herausfinden, aus welchen einzelnen Dateiversionen die freigegebene Version besteht, diese Version aus CVS auslesen und eine neue freizugebende Version erstellen. Mit drei Dateien, wie in unserem Beispiel, ist dies vielleicht noch möglich, aber bei einem Projekt, das eine bedeutend größere Anzahl von Dateien umfasst, bedeutet dies schon einen deutlichen Mehraufwand.

Die Lösung, die CVS bietet, besteht darin, bei Erreichen eines bestimmten Meilensteins die geänderte Version mit einem Tag zu versehen. Vor der Freigabe kennzeichnet Jane die Dateien mit dem Namen der Freigabe:

```
jane@dev:/home/jane/f-api > cvs tag rel-1_0
```

Wenn sie diese Freigabeversion später noch einmal benötigt, braucht sie nur den Befehl `cvs checkout` mit diesem Tag ausgeben:

```
jane@dev:/home/jane/f-api > cvs checkout -r rel-1_0 f-api
```

Und schon hat sie die Version 1.0 wieder. Sie wird im Verzeichnis `./f-api` abgelegt.

Das Konzept der Versionszweige ist etwas komplizierter, aber Sie können es sich wie strengere Tags vorstellen. Wenn Sie eine stärkere Trennung von den beiden Codebahnen haben möchten (die sich trotzdem noch auf dasselbe Projekt beziehen), würden Sie genaugenommen zwei separate Versionszweige erzeugen. Auf unser Beispiel bezogen gehen wir einmal davon aus, dass John und Jane zwei separate Versionen pflegen möchten – eine für die Entwicklung ihrer API und eine stabile Version – so wie bei der Entwicklung des Linux-Kernels.

Ein neuer Versionszweig wird mithilfe des Befehls `cvs tag -b` erzeugt. Um einen Entwicklungszweig mit der aktuellen Codebasis zu generieren, führt Jane folgenden Befehl aus:

```
jane@dev:/home/jane/f-api > cvs tag -b dev .
```

Der neue Versionszweig erhält den Namen `dev`. Um eine lokale Kopie dieses Zweiges zu erzeugen, braucht Jane ihn nur von CVS auslesen, wie sie es bei einer mit Tag gekennzeichneten Version tun würde:

```
jane@dev:/home/jane/f-api > cvs checkout -r dev f-api
```

Hier wird der Unterschied zwischen normalen, mit Tag gekennzeichneten Dateien und Dateien eines Versionszweiges deutlich. Alle Befehle, die Jane zukünftig ausführt, betreffen nur diesen Zweig und haben keine Auswirkung auf den stabilen Hauptzweig. Alle Daten werden im Zweig `dev` aktualisiert und hierhin weitergeleitet. Damit kann Jane im Entwicklungszweig arbeiten, ohne den Hauptzweig zu ändern. Aber Vorsicht! Die Pflege verschiedener Zweige kann sich zu einem verwalterischen Alptraum entwickeln. Wir empfehlen daher, nicht mehr als drei Zweige je Projekt zu verwenden.

Tipp: Wenn Sie sehen möchten, zu welchem Zweig eine Datei gehört, verwenden Sie `cvs status`. Die Ausgabe »Sticky Tag« gibt den Namen des Versionszweigs und die Versionsnummer an.

Häufig wird ein Abgleich zwischen dem neuen Zweig und dem Hauptzweig vorgenommen. Von Zeit zu Zeit möchte Jane die neuen Funktionen aus dem Entwicklungszweig wieder in den Hauptzweig übertragen. Dazu muss sie zunächst den Hauptzweig auslesen, und dann einen Diff für den Entwicklungszweig erstellen:

```
jane@dev:/home/jane/ > cvs checkout f-api
jane@dev:/home/jane/f-api/ > cvs update -j dev .
```

Die Option `-j` (join) des Befehls `cvs update` weist CVS an, die Unterschiede aus dem Entwicklungszweig mit der lokalen Kopie zu vermengen.

Tipp: Der Name des Hauptzweigs lautet immer HEAD. Jane könnte den Hauptzweig und den Entwicklungszweig mit dem Befehl `cvs update -j HEAD` zusammenfassen.

Nach Behebung möglicher Konflikte kann Jane ihre Kopie an das Hauptverzeichnis zurückschicken, und die Rückübertragung ist abgeschlossen.

Automatische Benachrichtigung

An diesem Punkt muss Jane wiederum ihre Kollegen über den Zusammenschluss informieren. Wie wir bereits erwähnt haben, kann CVS eine effektive Kommunikation im Team nicht ersetzen. Es kann sie aber unterstützen.

Der CVS-Server kann so eingerichtet werden, dass er bei jeder Übertragung zum Zentralarchiv eine bestimmte Aktion durchführt. Das Verschicken einer Email an einen Anschriftenlistenserver könnte eine solche Aktion sein. Indem Sie eine Anschriftenliste oder Aliasgruppe für jedes Projekt einrichten, können Sie Meldungen über Weiterleitungen einschließlich Protokolleinträge leicht an alle Mitglieder verbreiten.

Zu diesem Zweck muss eine der CVS-Verwaltungsdateien, nämlich die Datei `loginfo`, bearbeitet werden. Diese Datei legt fest, wie Informationen über das Zurückschicken der Daten gehandhabt werden. Sie können diese per Mail verschicken, in eine Datei protokollieren, sie in einer Datenbank speichern (oder ähnliches).

Sie können die administrativen Dateien direkt im Verzeichnis `CVSROOT` des Systems ändern. Wir empfehlen trotzdem, sie per CVS zu ändern. Auf diese Weise stehen Ihnen alle regulären Funktionen und das Sicherheitsnetz von CVS zur Verfügung. Um eine Sitzung zu starten, lesen Sie das Verzeichnis `CVSROOT` aus:

```
jane@dev:/home/jane/ > cvs checkout CVSROOT
```

Anschließend bearbeiten Sie die Datei `loginfo`. Der erste Teil der Zeile besteht aus einem regulären Ausdruck, der mit dem übertragenen Verzeichnis der Datei verglichen wird. Wenn eine Übereinstimmung gefunden wird, gibt der Rest der Zeile das Programm an, das aufgerufen wird. Das Programm sollte Übertragungsinformationen als Standardeingabe erwarten. Anstelle eines regulären Ausdrucks könnten Sie auch eine Zeile in der Datei verwenden, die mit `DEFAULT` beginnt. Diese Anweisung wird herangezogen, wenn kein übereinstimmendes Verzeichnis gefunden wird. Ein weiterer spezieller Befehl ist `ALL`, der immer ausgeführt wird. Zwei Beispiele solcher Zeilen:

```
^phpBook$ /cvs/loginfo_process_phpBook.sh
ALL /cvs/loginfo_process_all.sh
```

Das erste Shell-Skript `loginfo_process_phpBook.sh`, wird nur für das Archiv `phpBook` aufgerufen. Das zweite Skript `loginfo_process_all.sh` wird bei jeder Rückübertragung aufgerufen, ungeachtet des Archivs, da es das Schlüsselwort `ALL` enthält.

In das aufzurufende Programm können ein paar spezielle Variablen eingefügt werden, die ausführliche Informationen über die Rückübertragung angeben. Diese Variablen werden durch ein vorangestelltes Prozentzeichen (`%`) markiert, ähnlich der Formatdeklaration in `printf()`. Wenn mehr als eine Variable verwendet wird, müssen diese in geschweiften Klammern angegeben sein. Die folgende Tabelle zeigt die verfügbaren Variablen.

Variable	Bedeutung
s	Dateiname
V	Versionsnummer vor der Rückübertragung
v	Versionsnummer nach der Rückübertragung

Sobald Sie eine dieser Variablen verwenden, wird automatisch eine weitere Zeichenfolge mit dem Namen des CVS-Moduls vor der Variablen eingefügt.

Um eine Mail an John und Jane zu schicken, könnte folgende Zeile verwendet werden:

```
^f-api /bin/mail -s "CVS update: %s" -c john jane
```

Was ist jedoch, wenn die Rückübertragungsnachricht auch in einer Datei protokolliert werden soll? CVS überprüft nur den ersten Eintrag für ein Verzeichnis auf Übereinstimmung, so dass das Einfügen einer weiteren Zeile für `^f-api` nichts helfen würde. Zweifellos ließe sich dieses Problem durch ein Shell-Skript lösen. Aber was ist, wenn die Routine erweitert werden sollte, um später die Nachricht in einer Datenbank zu speichern? Listing 5.2 zeigt, wie Sie dieses Problem mit PHP lösen. In der Datei `loginfo` lautet der Aufruf folgendermaßen:

```
phpBook /usr/local/cvsroot/CVSROOT/Commit_Info.php3
➥"/usr/local/cvsroot/CVSROOT/logs/default.log" "CVS update %s" till tobias
```

Diese Zeile weist CVS an, das Skript `Commit_Info.php3` aufzurufen, wenn eine Datei oder ein Verzeichnis in das Archiv `phpBook` zurück übertragen werden soll. Das Skript wird mit mindestens drei Argumenten (in unserem Beispiel vier) aufgerufen:

- einem Dateinamen, in dem eine protokollarische Zusammenfassung dieser Rückübertragung gespeichert werden soll,
- der Themenzeile der Email-Nachricht,

- dem / den Empfänger / n. Mindestens ein Empfänger ist erforderlich. Wenn Sie die Nachricht an mehrere Empfänger verschicken möchten, fügen Sie, wie im vorhergehenden Beispiel gezeigt, einfach eine Adressliste hinzu.

```
// Check for correct number of arguments
if(count($argv) < 4)
{
    print("Usage: Commit_Info.php3 logfile subject to-address [to-address
...]\n");
    print("\n");
    print("A script to log and mail CVS commit messages.\n");
    print("    From Web Application Development with PHP\n");
    print("    Tobias Ratschiller and Till Gerken.
    ➥Copyright (c) 2000, New Riders Publishing\n");
    exit;
}

// First argument is the logfilename
$logfile = $argv[1];

// Second argument is the mail's subject
$subject = $argv[2];

// Initialize the body variable
$body = "";

// Get the commit message passed via stdin
$fp = fopen("php://stdin", "r") or die("Fatal Error: could not read from
➥stdin.");
while(!feof($fp))
{
    $body .=  fgets($fp, 64);
}
fclose($fp);

// Mail the message to all specified recipients
for($i=2; $i<count($argv); $i++)
{
    mail($argv[$i], $subject, $body);
}

// Log the message
$fp = fopen($logfile, "a") or die("Fatal Error: could not write logfile
➥($logfile) for writing");
fputs($fp, "$subject\n");
fputs($fp, "$body\n");
```

```
fputs($fp, "---------------------------------------------------------\n");
fclose($fp);
```

Listing 5.2: Meldung zur Protokollierung und zum Verschicken einer E-Mail

Warum verwenden wir überhaupt PHP für solch ein Kommandozeilenskript? Es wurde zwar ursprünglich als Webskriptsprache entworfen, hat sich aber soweit fort entwickelt, dass es sich auch für Aufgaben wie diese eignet. Wenn Sie PHP als unabhängigen Binärcode kompilieren, können Sie ihn als Skriptinterpreter wie Perl verwenden. Auf UNIX-Systemen können Sie in der ersten Zeile der Datei den Skriptinterpreter festlegen:

```
"#!/usr/bin/php
```

Wenn Sie das auführbare Bit hierfür setzen, können Sie diese Zeile wie jede andere ausführbare Datei in UNiX-Systemen aufrufen:

```
tobias@dev:/home/tobias/ > chmod +x script.php
tobias@dev:/home/tobias/ > ./script.php
```

Unter Windows können Sie dem Interpreter die Dateierweiterung `.php` zuordnen. Damit erreichen Sie das gleiche Ergebnis.

Das Skript zeigt zwei wichtige Konzepte, die in Kommandozeilenskripten häufig verwendet werden: der Zugriff auf Argumente, die an das Skript übergeben werden, und das Einlesen von Standardeingaben. PHP errichtet automatisch ein Array namens `$argv`, das als ersten Eintrag den Dateinamen des Skripts enthält und als weitere Einträge alle Argumente des Skripts. Dieses Verhalten entspricht jenen in C. Genau wie in C gibt es eine weitere Variable `$argc`, welche die Anzahl der Argumente angibt. Natürlich können Sie auch `count($argv)` verwenden. Wenn Sie ein PHP-Skript beispielsweise mit `php script.php3 foo` aufrufen, gibt `$argv[1]` das erste Argument `foo` an.

Der zweite beachtenswerte Teil des Programmcodes betrifft das Einlesen aus Standardeingaben. PHP 4.0 ermöglicht Ihnen, sogenannte *PHP-Streams* mit `fopen()` zu verwenden. Zur Annahme von Benutzereingaben in der Befehlszeile oder Einlesen von Daten, die von einem anderen Programm übergeben wurden, wird `php://stdin` verwendet. Das Skript verwendet diesen Befehl, um die Protokollmeldung zu lesen, die von CVS geliefert wird. Mit einer ähnlichen Funktion können Sie den Benutzer auffordern, Eingaben zu machen. Listing 5.3 zeigt eine solche Funktion.

```
function readln()
{
    // Initialize return value
    $ret = "";

    // Read from stdin until the user presses enter
```

```
    $fp = fopen("php://stdin", "r") or die("Fatal Error: could not read
    ➥from stdin");
    do
    {
        $char = fread($fp, 1);
        $ret .= $char;
    }
    while($char != "\n");

    return($ret);
}

print("Please enter your firstname:\n");
$firstname = readln();

print("You entered: $firstname\n");
```

Listing 5.3: Aufforderung zur Benutzereingabe mit PHP-Eingabe-/Ausgabe-Streams

Die anderen verfügbaren Streams heißen `php://stdout` und `php://sterr`. Sie bieten eine effektive, plattformunabhängige Alternative, um auf Standardein- und -ausgaben sowie die Standardfehlerbehandlungsroutinen zuzugreifen. Dies funktioniert sogar unter Windows 98 / NT.

Das Konzept der PHP-Streams wurde erst in PHP 4.0 eingeführt und ist daher in der Version 3.x nicht verfügbar. Auf UNIX-Systemen können Sie diese Beschränkung umgehen, indem Sie direkt auf die UNIX-Geräte `/dev/stdin`, `/dev/stdout` oder `/dev/stderr` zugreifen. Dies funktioniert natürlich nur auf Systemen mit gültigen Geräten. Für die meisten UNIX-Systeme trifft dies zu – auf Windows wiederum nicht.

Wenn Sie versuchen, mit CVS alle Ihre Dateien eines Website-Projektes zu verwalten, d.h. nicht nur den Quellcode sondern auch Bilder, Archive, Flashs usw., werden Sie bald feststellen, dass es mit binären Dateien einige Einschränkungen gibt. Einen Text-Diff zu erstellen, macht bei binären Daten keinen Sinn. In der Regel ist es für Sie uninteressant, welche Bits sich in einer Bilddatei geändert haben.

Wenn Sie alle Dateien als ASCII-Code behandeln, kann CVS tatsächlich Probleme hervorrufen, denn einige Zeichenfolgen werden von CVS als Schlüsselwörter betrachtet, z.B.

```
$Id: Basic_Web_Application_Strategies.xml,v 1.5 1999/12/15 15:49:55 tobias
Exp $
```

Diese werden bei der Übertragung zu Server durch die entsprechenden Langformen ersetzt, was bei binären Daten zu einigem Durcheinander führen könnte.

Sie können zwar manuell die Option -b angeben, wenn Sie mit cvs add eine Datei hinzufügen, aber dies wird bald umständlich zu verwenden und einzuhalten. Es wäre einfacher, wenn CVS den Binärtyp in einigen Dateierweiterungen erzwingen könnte. Glücklicherweise lässt sich dies einfach mit der Verwaltungsdatei cvswrappers bewerkstelligen. In dieser Datei können Sie Aktionen definieren, die Dateien bei jeder Rückübertragung und jedem Auslesen umwandeln. Sie ähnelt der Datei loginfo, da sie bei einer Rückübertragung aufgerufen wird, aber im Gegensatz zu den Anweisungen in der Datei loginfo kann sie Aktionen ausführen, welche die betreffende Datei ändern. Um Bilder und Archivdateien als Binärcode zu behandeln, könnte die Datei cvswrappers etwa so aussehen:

```
*.gif -kb *.jp[e]g -kb *.png -kb *.gz -kb *.ta -kb *.zip -kb
```

Hierdurch wird automatisch -kb an alle CVS-Befehle angehängt, die auf den angegebenen Filter passen. Die Option -k verhindert, dass CVS die Schlüsselworterweiterung verwendet. Der Schalter -b markiert eine Binärdatei und deaktiviert die Möglichkeit, Text-Diffs für diese Datei zu erstellen.

Manchmal müssen Sie im Web mit externen Kommandozeilenwerkzeugen kommunizieren. Viele UNIX-Programme erwarten die Eingabe über die Standardeingabe und geben die Ergebnisse der Berechnung über die Standardausgabe aus, z.B. sort. Die einfachste Möglichkeit, Eingaben weiterzuleiten und Ausgaben aus dem Programm zu lesen, bieten popen() und PHP-Streams. popen() öffnet eine einseitig gerichtete Pipe zu einem Programm, über welche Sie Daten an seine Standardeingabe übertragen können. Wie Sie bereits wissen, können Sie fopen() verwenden, um die Standardeingabe zu lesen, in welche das externen Programm schreibt. Ein einfaches Beispiel hierfür sieht folgendermaßen aus:

```
// Open a writing pipe to the sort command
$fpout = popen("sort", "w");

// Open stdin with PHP-Streams
$fpin = fopen("php://stdin", "r");

// Output some characters to sort
fputs($fpout, "a\n");
fputs($fpout, "c\n");
fputs($fpout, "b\n");

// Close the pipe to sort - sort will now process the input
// and write it to stdout
pclose($fpout);

// Sort's stdout is stdin for us - so read it until feof().
while($c = fread($fpin, 1))
```

```
{
    print($c);
}
```

CVS-Kurzreferenz

CVS hat zwar nur etwa 25 Hauptbefehle, aber eine Vielzahl an Optionen und noch mehr mögliche Kombinationen. In der Praxis werden Sie nur einen kleinen Teil der Befehle benötigen. Die folgende Tabelle gibt Ihnen einen kurzen Überblick, den Sie als Gedächtnisstütze für die grundlegenden Befehle verwenden können.

Befehl	Beispiel	Beschreibung
cvs login	cvs login	Dient zur Anmeldung auf dem CVS-Server. Dieser fragt nach Ihrem Passwort.
cvs checkout	cvs checkout phpBook	Liest ein Modul aus dem CVS-Zentralarchiv aus. Wird in der Regel zum Auslesen eines Moduls verwendet (gesamtes Verzeichnis unterhalb der obersten Ebene).
cvs update	cvs update TOC.xml	Aktualisiert Ihre lokale Kopie einer Datei oder eines Verzeichnisses.
cvs commit	cvs commit TOC.xml	Sendet Ihre lokale Version der Dateien oder Verzeichnisse an den CVS-Server.
cvs add	cvs add TOC.xml	Fügt eine neue Datei oder Verzeichnis auf dem CVS-Server hinzu. Die Aktion wird bei der nächsten Übertragung durchgeführt.
cvs remove	cvs remove TOC.xml	Löscht einen Datei oder ein Verzeichnis auf dem CVS-Server. Die Aktion wird bei der nächsten Übertragung durchgeführt. Beachten Sie, dass dieser Befehl eine Datei nicht tatsächlich löscht. Sie können immer noch auf ältere Versionen zugreifen.
cvs status	cvs status TOC.xml	Zeigt den Änderungsstatus Ihrer lokalen Version an.
cvs diff	cvs diff TOC.xml	Zeigt die Unterschiede zwischen zwei Versionen einer Datei. Standardmäßig werden die Unterschiede zwischen der lokalen Kopie und der Version im Zentralarchiv angezeigt.
cvs log	cvs log TOC.xml	Zeigt die CVS-Protokollmeldung für alle Versionen einer Datei an.

5.4 Dreistufige Anwendungen

Bei größeren Entwicklungsteams sind effektive Mittel zur Organisierung eines Projekts lebenswichtig. Bisher haben wir Ihnen Verfahren gezeigt, wie Sie Layout und Code trennen, Verzeichnisstrukturen organisieren und ein Versionskontrollsystem zur Verwaltung des Quellcodes einsetzen. Sobald die Sprache auf verteilte Entwicklung, verschiedene Entwicklungsphasen und Geschäftsprozesse kommt, erscheinen Marketingfachleute auf der Bildfläche und schreien nach mehrstufigen Anwendungen. Was soll das Ganze?

5.4.1 Herkömmliches Client/Server-Modell

In der Vergangenheit war eine Nicht-Webanwendung verantwortlich für die Abwicklung aller Angelegenheiten, beginnend bei der Benutzereingabe über die Anwendungslogik bis zur Datenspeicherung. Diese drei Einheiten waren miteinander verwoben, was es schwierig, wenn nicht gar unmöglich machte, eine von ihnen ohne Auswirkungen auf die anderen zu ändern. Wenn Sie solch eine Anwendung mehreren Benutzern zur Verfügung stellen wollten, hatten Sie ein Problem. Was ist, wenn sich das Datenformat ändert? Was passiert, wenn Sie die Programmlogik ändern müssen? Sie müssten alle Benutzer mit einer neuen lokalen Kopie der Anwendung versehen, was bei größeren Systemen unmöglich ist.

Mit der Einführung der objektorientierten Entwicklung in bezug auf Analyse, Design und Programmierprinzipien wurden Geschäftsanwendungen modular. Benutzerschnittstellen wurden üblicherweise auf Arbeitsplatzrechnern eingesetzt, während Daten und Programmlogik auf einem Mainframeserver untergebracht waren. Der Begriff *Client/Server* wurde verwendet, um die Trennung der Schichten oder Stufen zu beschreiben. Eine zweistufige Client/Server-Architektur bietet grundsätzlich die Möglichkeit, Aufgaben zu trennen. Der Client bzw. die erste Stufe ist für die Darstellung der Daten für den Benutzer (Benutzerschnittstelle) verantwortlich, während der Server bzw. die zweite Stufe für die Bereitstellung von Datendiensten für den Client und die Verwaltung der Programmlogik zuständig ist. So weit, so gut. Aber wie Sie sehen, kombiniert das zweistufige Modell immer noch zwei verschiedene Konzepte in der zweiten Stufe (dem Server): *Datendienste* und *Programmlogik*. Die Programmlogik bildet das Herz des Programms, das für die Datenüberprüfung, Verarbeitungsregeln usw. verantwortlich ist.

Die Datendienste verwalten die Rohdaten der Anwendung. Häufig bestehen sie aus einem relationalen Datenbankverwaltungssystem (RDBMS) oder einem schon vorhandenen Mainframesystem, in das die Firma bereits viel Zeit und Geld investiert hat. Die Vermischung beider Konzepte in einer zweistufigen Anwendung führt zu Problemen der Skalierbarkeit, Wiederverwendung und Datenpflege. Nehmen wir als Beispiel den kürzlich eingeführten

Euro als offizielle Währung der Europäischen Gemeinschaft. Wenn hier die Programmlogik nicht klar von den Daten getrennt worden wäre, müssten beide Stufen geändert werden. Fest in der Datenüberprüfung einprogrammierte Kurswerte müssten genauso geändert werden wie das Datenspeicherformat der Währungsbeträge.

Mithilfe eines mehrstufigen Ansatzes können Sie alle drei Konzepte klar voneinander trennen.

5.4.2 PHP und mehrstufige Anwendungen

Das Konzept der mehrstufigen Systeme errang in der frühen 90er Jahren Beliebtheit und etabliert sich inzwischen zunehmend als Firmenprogrammarchitektur des ausgehenden 20sten und beginnenden 21sten Jahrhunderts. Eine mehrstufige (häufig auch als dreistufig bezeichnete) Architektur bietet eine größere Skalierbarkeit für Anwendungen, eine geringere Datenpflege und bessere Wiederverwendbarkeit von Modulen. Eine dreistufige Architektur bietet die Möglichkeit einer technologieunabhängigen Methode zur Einrichtung von Client/Server-Anwendungen für Hersteller, die Standardschnittstellen einbeziehen und für jede logische Schicht Dienste bereitstellen. Betrachten Sie das dreistufige Modell in Abbildung 5.5 – kommt es Ihnen als Webanwendungsentwickler nicht bekannt vor? In der ersten Stufe ist ein dünner Client. Übertragen auf die Welt der Webanwendungen wäre dies der Browser. Die mittlere Stufe (Anwendungsserver) entspricht offensichtlich PHP (und dem Webserver als Hostrechner), während die dritte Stufe aus einem RDBMS besteht.

Wenn Sie PHP verwenden, arbeiten Sie mit einem einfachen dreistufigen Modell, und zwar auf so kohärente Weise, dass Sie es kaum bemerken werden. Etwas Besonderes war und ist die Verwendung von PHP als Anwendungsserver. Bis zur Version 4.0 war PHP nicht darauf ausgerichtet, auf Softwareobjekte Dritter zuzugreifen, und daher nicht das beste Werkzeug für Geschäftsszenarien mit ähnlichen Anforderungen. Dies hat sich geändert und wir halten es für eine der größten Errungenschaften von PHP. Mit der aktuellen Version können Sie nicht nur COM-Bedienelemente in WIN32-Systemen verwenden, sondern auch Java-Befehle auf beliebigen Systemen mit einer Java-Virtual-Machine (JVM) ausführen.

Es gibt drei Standardarchitekturen für solch eine firmenweite Wiederverwendung von Objekten: (D)COM von Microsoft, JavaBeans/RMI von Sun Microsystems und CORBA von der Object-Management-Group. Aufgrund der weit verbreiteten Verwendung von Windows-Systemen wird COM am häufigsten eingesetzt (Einzelheiten über die Verwendung von PHP und COM finden Sie im nächsten Abschnitt). Unternehmen mit heterogeneren Umgebungen bevorzugen allerdings Java oder CORBA, da sie mehr Plattformen unterstützen.

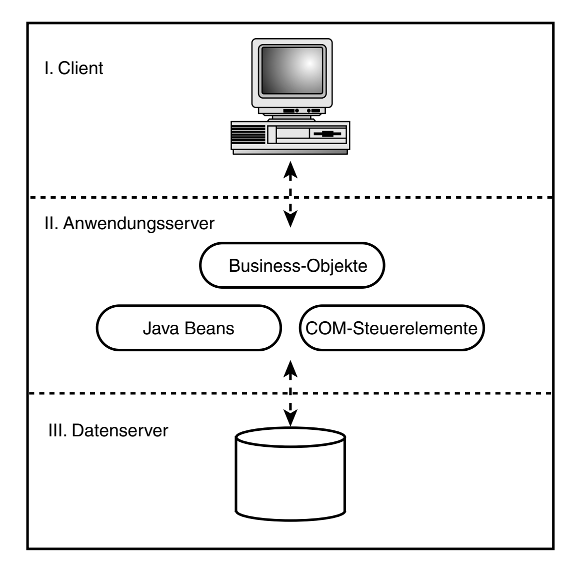

Abbildung 5.5: Schichten mehrstufiger Anwendungen

Für den Zugriff auf SAP R/3-Systeme steht beispielsweise JavaBeans (oder alternativ COM-Objekte) mit ihrer Business-API (BAPI)-Schnittstelle zur Verfügung, über welche Sie das SAP Business Framework in Ihre Anwendungsserverschicht integrieren können. Überlegen Sie, welche Möglichkeit dies für ein E-Commerce-System eröffnen könnte. Sie könnten einen Online-Shop des Benutzers mit dem Warehouse-Programm der Firma kombinieren.

5.4.3 PHP und COM

Das Component-Object-Model (COM) ist eine von Microsoft entwickelte Softwarearchitektur, die eine Unterteilung von Programmteilen in binäre Module ermöglicht. COM definiert einen Standard für das Zusammenspiel der einzelnen Module und spezielle Funktionen, die jedes Modul haben muss. Es hängt von keiner Programmiersprache ab; Microsoft versucht sogar, COM als offenen IETF(Internet Engineering Task Force)-Standard zu etablieren. Auf COM-Objekte kann mit kompatiblen Anwendungen und Programmiersprachen zugegriffen werden, z.B. über Visual Basic, Delphi oder PHP.

COM beschäftigt sich mit folgender Frage: Wie kann ein System aufgebaut werden, so dass binäre Programmkomponenten von verschiedenen Herstellern zusammenarbeiten können? Microsofts erste Antwort auf diese Frage, nämlich das Konzept der DLL, scheiterte kläglich. Aufgrund der fehlenden Versionskennungen in den DLLs können auf demselben System keine zwei DLLs verwendet werden, welche dieselbe Schnittstelle benutzen. Betrachten

wir ein Beispiel. Angenommen, Sie haben ein serverüberwachendes Programm, dass in regelmäßigen Abständen überprüft, ob ein Server noch arbeitet, und die gesammelten Daten protokolliert. Daneben haben Sie einen Frontend mit schönen Statistiken und ein wenig Programmlogik, der entsprechend reagiert, wenn der Server nicht mehr erreichbar ist. Außerdem könnte die Software noch eine DLL als Backend haben, welche die tatsächliche Überwachung übernimmt. Sagen wir, dass `ServerSpy.dll` eine Funktion `is_reachable()` enthält, die als Argument den Servernamen akzeptiert und `true` zurückgibt, wenn der Server erreichbar ist, ansonsten `false`.

Bisher haben wir noch nicht definiert, welche Art von Server wir überwachen wollten. Es könnte ein Webserver, Mailserver, Nameserver oder jedes andere System sein. Die Funktionsdeklaration selbst und der Backend müssten nicht geändert werden. Der einzige Teil, der angepasst werden müsste, wäre die Implementierung der Funktion `is_reachable()`.

Mit DLLs können Sie genau einen Überwachungsdienst auf Ihrem System installieren: entweder einen Webserverspion oder einen Mailserverspion usw. Sobald Sie eine weitere DLL installieren, würde diese die vorherige überschreiben.

Der Kunde könnte in Ihrer Anwendung nicht zwischen verschiedenen Überwachungsdiensten auswählen. Sie könnten auch nicht verschiedene Versionen eines Dienstes auf dem System verwenden. Wenn ein anderer Softwarehersteller eine bessere Version von `ServerSpy.dll` schreiben und diese an Ihrem Kunden verkaufen würde, würde diese wiederum Ihre DLL-Version überschreiben. Sofern die neue Funktion genauso wie die alte funktioniert, ist alles großartig, anderenfalls wird Ihr Programm den Dienst einstellen.

Ein gutes Komponentenmodell versucht diese Probleme zu lösen. COM bietet die Möglichkeit, Komponenten über eine globale, eindeutige ID (GUID) zu kennzeichnen, so dass Sie viele Dienste gleichzeitig verwenden können. Das Betriebssystem kennt alle installierten Komponenten, und Ihr Programm könnte es in der Kategorie »Serverspion-Überwachungsdienste« abfragen. Außerdem verfügt COM über eine Versionskontrolle, so dass Sie verschiedene Versionen einer Komponente auf dem System einsetzen können, ohne dass sich diese gegenseitig behindern.

Schließlich verfügt es über eine Komponenten-Introspektion, mit der Sie die Methoden und Eigenschaften einer Komponente einsehen können.

Möglicherweise haben Sie schon einmal von ActiveX-Steuerelementen gehört. ActiveX-Elemente, die auch als OLE-Elemente oder – dank Microsofts kreativer Marketingabteilung – als OCX verbreitet wurden, sind nichts anderes als eine Anwendung der COM-Komponenten. Sie haben einen reduzierten Satz an COM-Schnittstellen, so dass sie kleiner und besser für Netzwerke mit hoher Latenzzeit geeignet sind. Sie können die ActiveX-Elemente nahezu genauso verwenden wie die von ausgestatteten COM-Elementen.

Des weiteren gibt es noch DCOM, das Distributed-Component-Object-Model-Protokoll, das COM-Komponenten für die Welt der verteilten Netzwerkanwendungen bereitstellt. Im Prinzip handelt es sich hierbei um ein Protokoll für objektorientierte Prozedurfernaufrufe, die für verteilte modulbasierte Systeme nützlich sind. Auf Win32-Systemen funktioniert PHP 4.0 sowohl mit COM als auch mit DCOM. Auf anderen Systemen, auf denen COM verfügbar wäre (wie etwa Apple oder Solaris), funktioniert es leider noch nicht.

Der Aufruf der (D)COM-Elemente ist mit PHP 4.0 recht einfach. `COM` ist ein reservierter Klassenname, von dem Sie eine Instanz erstellen können, indem Sie den Namen des Steuerelements an den Konstruktor übergeben. Damit erhalten Sie eine vollständige Instanz des `COM`-Objekts, mit dem Sie Funktionen ausführen oder Eigenschaften setzen bzw. lesen können, so als ob es sich um eine PHP-Klasse handelte. Bei Parametern, die sich nicht mit der PHP-Syntax ausdrücken lassen (weil sie beispielsweise einen Punkt oder ein anderes, in PHP ungültiges Zeichen enthalten), können Sie sich mithilfe der folgenden Funktionen behelfen:

- `bool com_set(class com_object, string property_name, string property_value)`
- Ordnet einem Parameter des COM-Objekts, das in `com_object` instantiiert wurde, einen Wert zu. Aliase für diese Funktion sind `com_propset()` und `com_propput()`. Gibt bei erfolgreicher Zuordnung `true` zurück, ansonsten `false`.

  `mixed com_get(class com_object, string property_name)`
- Gibt den Wert eines Parameters des COM-Objekts aus, das in `com_object` instantiiert wurde. Alias für diese Funktion ist `com_propget()`. Gibt entweder den Wert des Parameters zurück oder bei einem Fehler `false`.

Der Quellcode in Listing 5.4 zeigt die grundlegende Verwendung der COM-Funktion. Mithilfe der Komponente `Softwing.EDConverter`, einem Währungsumwandlungsprogramm, wird eine Instanz der Klasse `COM` erzeugt. Anschließend wird die Mitgliedsmethode `Triangulate()` aufgerufen und das Ergebnis angezeigt. Viel einfacher geht es nicht...

```
$amount = 1000;          // Amount to be converted
$curr_from = "DEM";      // ISO currency symbol of original currency
$curr_to = "ITL";        // Symbol of the target currency

// Instantiate new COM object
$conv = new COM("Softwing.EDConverter") or die("Unable to instantiate
➥Euro-Converter");

// Execute a component method on the COM object's instance
```

```
$ret = $conv->Triangulate(10000, "ITL", "DEM") or die("Exception triggered
➥by Triangulate() on line ".__LINE__);

// Print the result
print($ret);
```

Listing 5.4: Ein einfaches COM-Beispiel

Um auf einem entfernten System eine Instanz einer DCOM-Komponente zu erzeugen, übergeben Sie den Hostnamen als zweites Argument an den Konstruktor. Ein Beispiel:

```
$comp = new COM ("My.Component", "remote.server.com");
```

In PHP 3.0 können Sie die Klasse `COM` nicht verwenden. Statt dessen müssen Sie die Funktionen `com_load()`, `com_invoke()`, `com_set()` und `com_get()` benutzen. Unser Beispiel sähe so aus wie in Listing 5.5.

```
$amount = 1000;          // Amount to be converted
$curr_from = "DEM";      // ISO currency symbol of original currency
$curr_to = "ITL";        // Symbol of the target currency

// Instantiate new COM object
$conv = com_load("Softwing.EDConverter") or die("Unable to instantiate
➥Euro-Converter");

// Execute a component method on the COM object's instance
$ret = com_invoke($conv, "Triangulate", "10000", "ITL", "DEM") or
➥die("Exception triggered by Triangulate() on line ".__LINE__);

// Print the result
print($ret);
```

Listing 5.5: Ein einfaches COM-Beispiel mit PHP 3.0

Diese Funktionen (die in PHP 4.0 nicht verfügbar sind) haben folgende Syntax:

- `int com_load(String component_name)`
  Generiert eine Instanz der Komponente `COM` und erzeugt eine Referenz darauf. Der zurückgegebene ganzzahlige Wert muss in den folgenden `com_*()`-Aufrufen verwendet werden. Gibt bei einem Fehler `false` zurück und emittiert eine Warnung.
- `mixed com_invoke(Int com_identifier, String function_name[, mixed argument1[,...]])`
  Ruft die Methode einer `COM`-Komponente auf und gibt den Rückgabewert der Methode zurück. Das erste Argument der Funktion ist ein gültiger `COM`-Bezeichner, der mit `com_load()` erzeugt wurde. Das zweite Argument muss

der Name einer Komponentenmethode sein. Optional können Sie in das dritte und alle folgenden Argumente die Argumente für die aufgerufene Methode eingeben. Bei einem Fehler gibt die Funktion `false` zurück.

- `bool com_set(Int com_identifier, String property_name, String property_value)`
  Weist einem Parameter des `COM`-Objekts, das in `com_object` instantiiert wurde, einen Wert zu. Aliase für diese Funktion sind `com_propset()` und `com_propput()`. Gibt bei erfolgreicher Zuweisung `true` zurück, bei einem Fehler `false`.
- `mixed com_get(Int com_identifier, String property_name)`
  Gibt den Wert eines Parameters des `COM`-Objekts zurück, das in `com_object` instantiiert wurde. Alias für diese Funktion ist `com_propget()`. Gibt den Wert der Eigenschaft zurück oder bei einem Fehler `false`.

5.4.4 PHP und Java

Java, das in vielen Firmen für Softwareentwickler die »Sprache des Tages« ist, wird zunehmend in allen Entwicklungsbereichen eingesetzt. Seit der Version 4.0 kann PHP so kompiliert werden, dass es den Aufruf reiner Java-Funktionen unterstützt, so dass Sie die Java-Komponenten der Firma in einer mehrstufigen Umgebung verwenden können.

Die Realisierung der Java-Unterstützung ist nicht schwierig. Folgende Punkte sollten Sie bedenken:

- Zunächst müssen Sie eine Java-Virtual-Machine (JVM) auf Ihrem System installieren. Wir haben hierfür das frei verfügbare Open-Source Kaffe 1.0.5 verwendet.
- PHP muss mit APXS als DSO kompiliert werden, wodurch es als gemeinsam genutztes Modul auf dem Apache-Server geladen wird. Die Befehle hierfür finden Sie in der Datei `INSTALL.DSO` der PHP-Distribution.

Weitere Anweisungen können Sie in der Datei `README` im Verzeichnis `ext/java` nachlesen.

Sobald Sie in PHP eine Java-Unterstützung haben, ist die Syntax vergleichbar mit jener für den Aufruf der `COM`-Komponenten. Tatsächlich nutzen beide eine erweiterte interne Zend-Funktion, die sogenannte Objektüberladung. Überladung bedeutet im Bereich der Softwareentwicklung, dass ein Konstrukt je nach Kontext unterschiedlich reagieren kann. Funktionsüberladung bedeutet demnach, dass eine Funktion je nach Anordnung oder Typ der Argumente unterschiedlich sein kann.

In C++ gibt es beispielsweise zwei Funktionen deklariert, die in etwa folgendermaßen aussehen:

```
void add(int Left, int Right);
void add(double Left, double Right);
```

Je nachdem, ob Sie einen int- oder double-Wert an add() übergeben, wird die eine oder die andere Funktion aufgerufen. In der Regel versuchen Programmierer jedoch, solche Mehrdeutigkeiten zu vermeiden, da sie bei der Fehlersuche zu einem Alptraum werden können.

Die Zend-Engine ermöglicht intern eine ähnliche Funktionsüberladung, und zwar zur Realisierung der Objektüberladung. Je nach Kontext, z.B. des Klassennamens, kann ein Objekt ganz unterschiedliche Bedeutungen haben: Es kann eine normale, vom Benutzer definierte, Klasse sein oder aber auch etwas sehr viel Anspruchvolleres, wie etwa eine COM- oder Java-Schnittstelle.

Ähnlich wie bei COM wird ein neues Java-Objekt erstellt, indem eine Instanz der überladenen Java-Klasse erstellt wird. Das Argument des Konstruktors gibt die zu verwendende Java-Klasse an:

```
$system = new Java ("java.lang.System");
```

Im zurückgegebenen Objekt können Funktionen genauso eingesetzt werden wie COM-Objekte.

Um Parameter zu lesen und zu schreiben müssen Sie die Hilfsfunktionen getProperty() und setProperty() verwenden (im Gegensatz zum COM-Modul, bei welchem direkt auf Parameter zugegriffen werden kann).

```
$system = new Java ("java.lang.System");
printf("Java version = %s\n", $system->getProperty("java.version"));
```

5.5 Zusammenfassung

In diesem Kapitel haben Sie die grundlegenden Strategien kennen gelernt, die in allen Webanwendungen üblich sind. Sie wissen, wie Sie mit Benutzereingaben richtig umgehen, wie Sie Formulare auswerten und ihren Inhalt überprüfen können. Sie können Schablonen verwenden, um Programmcode und Layout zu trennen. Darüber hinaus wissen Sie, wie Sie die Bemühungen Ihres Teams koordinieren und was Sie tun müssen, wenn mehr als eine Person an einer Datei arbeitet. Wir haben die Vorteile von Versionskontrollsystemen erörtert, und Sie können Ihre Projekte auf der Festplatte klar strukturieren. Sie können das flexible CVS-System intensiv nutzen. Zum Schluss sind wir auf mehrstufige Anwendungen eingegangen, die auch eine Implementierung von Schnittstellen für Fremdsprachen ermöglichen.

Ausgestattet mit diesem Wissen sind Sie nun in der Lage, Anwendungen der Spitzentechnologie zu konzipieren und zu implementieren, auf die wir im weiteren Verlauf eingehen werden.

6 Datenbankzugriff mit PHP

Wenn Sie mit den Werkzeugen des Zimmermanns spielen,
besteht die Möglichkeit, dass Sie sich in die Hand schneiden.

Datenbanken gehören zum täglichen Handwerkszeug eines Webentwicklers. Dieser sollte SQL zumindest genauso gut kennen wie PHP. Bei den meisten Datenmodellen von Webanwendungen ist eine relationale Datenbank im Spiel. Während unerfahrene Benutzer möglicherweise versuchen, den Overhead, den SQL und ein relationales Datenbankverwaltungssystem (RDBMS) mit sich bringen, zu umgehen, werden erfahrene Entwickler seine Funktionen schätzen. Alles, was etwas anspruchvoller ist, z.B. gleichzeitige Zugriffe, Such- und Sortierfunktionen oder Beziehungen zwischen verschiedenen Datensätzen, kann schnell zum Alptraum werden, wenn Sie herkömmliche Speichermethoden, wie zweidimensionale Dateien oder Arrays, verwenden. Datenbanken sind auf die effiziente Organisation und Abfrage von Daten spezialisiert und in den meisten Fällen gibt es keine Notwendigkeit, diese Funktionen in Pseudo-Datenbanksystemen zu imitieren.

Dieses Kapitel zeigt Ihnen, wie Sie mit PHPLib auf Datenbanken zugreifen, und beschreibt zwei der zahlreichen Funktionen dieser Bibliothek: Authentisierung von Benutzern und Verwaltung von Zugriffsrechten.

6.1 PHPLib: Die PHP-Basisbibliothek

Wie wir im Buch bereits erwähnt haben, kann PHPLib bei der täglichen Programmierroutine einen beträchtlichen Teil der Arbeit abnehmen. Einige Konzepte tauchen bei der Entwicklung von Webanwendungen immer wieder auf: Sitzungsverwaltung, Genehmigung, Trennung von Layout und Programmcode. PHPLib besteht aus einem festen Satz von Objekten, die Lösungen für diese unerlässlichen Aufgaben bieten.

Für viele Programmierer wirkt PHPLib einschüchternd, wenn sie die Dokumentation und die Beispiele betrachten. Tatsächlich besteht die Bibliothek aus einem recht komplexen Satz von Klassen, und die einzelnen Objekte hängen auf nicht ganz einfache Weise voneinander ab. Sobald Sie jedoch die Installation erfolgreich gemeistert und Ihre eigenen Basisklassen geschrieben haben, werden Sie feststellen, dass es gar nicht so schwierig zu verwenden ist und die Beispiele direkt und leicht zu verstehen sind.

PHPLib können Sie unter http://phplib.netuse.de/ herunterladen. Seine Dokumentation enthält ausführliche Installationsanweisungen, so dass wir hier nicht weiter darauf eingehen. Wir gehen davon aus, dass Sie eine korrekt eingerichtete PHPLib-Umgebung haben, bei welcher jedem Skript automatisch prepend.php3 vorangestellt wird (diese Einstellung wird in der Dokumentation empfohlen). Wir geben hier auch keine Beschreibung der einzelnen Funktionen und Parameter, da diese in der Dokumentation nachzulesen sind. Statt dessen zeigen wir den großen Rahmen auf und erläutern PHPLib mit unseren eigenen Worten.

6.1.1 Entstehung

Die erste Version von PHPLib wurde 1998 von Boris Erdmann und Kristian Koehntopp entwickelt. Während sie in Kiel bei einem Internetdienstanbieter an einem großen Projekt arbeiteten, stellten sie fest, dass sie immer wieder dieselben Prozeduren programmierten. Sie fanden die Art, diese Probleme zu lösen, nicht sehr zufriedenstellend.

Sie brauchten beispielsweise eine Anmeldeprozedur, die nicht auf der HTTP-Authentisierung basierte, da dieses Genehmigungsverfahren weder sicher noch benutzerfreundlich ist. Für eine korrekte Genehmigung brauchten sie eine funktionierende Sitzungsverwaltung (wie wir bereits in Kapitel 4 »Webanwendungskonzepte« hervorgehoben haben). Also begannen sie, eine objektorientierte Bibliothek für die Sitzungsverwaltung und -genehmigung zu entwickeln, die sich auf einem Konzept für die Sitzungsverwaltung von Karl-Heinz Wild stützt. Wie bei vielen Open-Source-Projekten kamen im Laufe der Zeit immer mehr Leute hinzu, und das Projekt wuchs schnell. Inzwischen verfügt PHPLib über Module für viele Aspekte der Genehmigung- und Sitzungsverwaltung, und besitzt verschiedene Verfahren, um HTML-Eingabeformulare, Tabellen und Baumstrukturen zu erstellen.

6.1.2 Vorteile und Nachteile

Wie wir in Kapitel 1 »Entwicklungskonzepte« erläutert haben, sollten Sie Ihre Werkzeuge sorgfältig auswählen. PHPLib eignet sich am besten für Projekte, die mehr als zwei Tage Entwicklung benötigen. Bei der ersten Verwendung der Bibliothek müssen Sie jedoch zunächst mit Mehrarbeit rechnen: Sie brauchen Zeit, um die Dokumentation zu lesen, die Konzepte zu verstehen und Ihre Fehler zu beheben.

PHPLib scheint ideal für Projekte mit mehr als einem Softwareentwickler zu sein. Es zwingt Programmierer, ähnliche Schnittstellen zu verwenden, und fördert objektorientierte Programmierung, die zumindest die Struktur einer Anwendung verbessert. Da PHPLib zudem ein tieferes Verständnis von PHP und Webanwendungen im allgemeinen voraussetzt, befinden sich die Entwickler im Team auf einem vergleichbaren Wissensniveau.

PHPLib ist in reinem PHP geschrieben, wodurch es nicht ganz so schnell ist wie eine C-Erweiterung von PHP. Auf der anderen Seite wird es dadurch flexibler. Die Bibliotheken sind als Klassen definiert; dadurch lassen sie sich leicht an Ihre Bedürfnisse anpassen. Tatsächlich werden Sie einige Änderungen vornehmen müssen: PHPLib ist kein Produkt, das Sie einfach auspacken und sofort benutzen können. Sie sollten sich darüber im Klaren sein, dass Sie einige Funktionen selbst bereitstellen müssen.

Die einzelnen PHPLib-Klassen sind auf recht komplexe Weise miteinander verwoben; so können Sie beispielsweise Sitzungsverwaltungsfunktionen nicht ohne die Datenbank-Abstraktionsschicht der Bibliothek verwenden. Wenn Sie nur eine Sitzungsverwaltung benötigen, empfiehlt es sich möglicherweise, eher die Sessionfunktion von PHP 4.0 oder unseren PHP 3.0-Backport zu verwenden. Sie sollten trotzdem weiterlesen: PHPLib verfügt über einige Funktionen, die Ihnen das Leben definitiv einfacher machen können.

6.1.3 Wichtige Dateien

In der PHPLib-Distribution gibt es zwei Dateien, die Sie wahrscheinlicherweise ändern müssen: `local.inc` und `prepend.php3`.

Die Datei `prepend.php3` lädt die Dateien, die auf allen Seiten verfügbar sein sollen, die PHPLib verwenden. Standardmäßig lädt es den Datenbank-Backend für MySQL und verwendet zur Sitzungsverwaltung des SQL-Speicherbehälters. Um beispielsweise von MySQL auf Postgres zu wechseln, müssten Sie folgende Zeile verändern:

```
require($_PHPLIB["libdir"] . "db_mysql.inc");  /* Change this to match your
➥database. */
```

und anstelle von `db_mysql.inc` fügen Sie `db_pgsql.inc` ein.

Wenn Sie andere Klassen aus PHPLib verwenden möchten (beispielsweise die Klasse `Template`), sollten Sie diese besser in `prepend.php3` einfügen.

Die Datei `local.inc` befindet sich in dem Verzeichnis, in welchem die Anpassung von PHPLib stattfindet.

6.1.4 Anpassung von PHPLib

In den meisten Fällen werden die Basisklassen von PHPLib nicht direkt verwendet. Statt dessen verwenden Sie Ihre eigenen, abgewandelten Klassen, welche auf Ihre Umgebung abgestimmt sind. Standardmäßig werden bereits einige Klassen mit PHPLib mitgeliefert, die wir später in unserem Beispiel verwenden werden. Sie machen einige Annahmen über Ihr System, z.B. dass Sie MySQL verwenden. Wenn diese Annahmen auf Ihre Anwendungen nicht zutreffen, sollten Sie die entsprechenden Änderungen in der Datei `local.inc`

vornehmen. Wir empfehlen, ein neues Objekt als Erweiterung zu `DB_Sql` zu erzeugen. Damit vermeiden Sie, diese Konfigurationsvariable in jeder Anwendung setzten zu müssen. Statt dessen definieren Sie sie einmal in einer Klasse.

Sie sollten am besten `local.inc` an Ihre eigenen Bedürfnisse anpassen, und zumindest den Namen der Klasse `Example_Session` ändern. Denn dieser Name wird als Bezeichnung für die Sitzungs-Cookies verwendet und in der URL im GET-Mode übergeben – und es sieht nicht sehr professionell aus, wenn in Ihre URL `Example_Session` steht.

6.2 Datenbankabstrahierung

Eine Datenbank-Abstrahierungsschicht ist eine API, die eine Reihe von Funktionen bereitstellt, mit denen Sie eine Vielzahl von Datenbanken unabhängig von der jeweiligen Implementierung verwalten können. Durch Ändern des Backends der Datenbank-Abstrahierungsschicht können Sie leicht von MySQL auf Oracle umschalten. Perls DBI (DataBase Interface) ist eine solche Schicht, und eine der bekanntesten Funktionen von PHPLib ist seine Datenbank-Abstrahierungsschicht, die in der Klasse `DB_Sql class` organisiert ist.

6.2.1 Portierbarkeit

Für einen professionellen Webanwendungsprogrammierer kann die Datenbankabstrahierung nützlich und wichtig sein. Die Basis aller Anwendungen ist das Datenmodell, ein Satz von Datenstrukturen für allgemeine Zwecke, die oft in Datenbanken enthalten sind. PHP unterstützt zwar eine große Anzahl von Datenbanken, aber jede Datenbank hat eine andere Anwendungsprogrammschnittstellte (API). Wenn Sie diese datenbankeigenen APIs verwenden, ist eine vom Betriebssystem und der Datenbank unabhängige Programmierung unmöglich. Wenn Sie eine Anwendung von MySQL auf Oracle portieren wollen, müssen Sie mit einer großen Anzahl von Anpassungen rechnen, es sei denn, Sie verwenden eine Abstrahierungsschicht, wie PHPLib. Tabelle 6.1 zeigt, wie Datenbankschnittstellen von System zu System variieren können.

Beschreibung	MySQL	Oracle 7
Verbinden	`mysql_connect()`	`ora_logon()`
Abfragen	`mysql_query()` oder `mysql_db_query()`	`ora_parse()`, dann `ora_exec()`

Tab. 6.1: APIs für den Zugriff auf MySQL und Oracle

Beschreibung	MySQL	Oracle 7
Nächste Ergebniszeile einlesen	oder `mysql_fetch_array()`	Arbeitet mit Offsets: `ora_columnname()`, `ora_getcolumn()`
Anzahl der Zeilen in einem Ergebnis	`mysql_num_rows()`	Nicht möglich, da Oracle bereits Zeilen zurückgibt, bevor die Gesamtanzahl der Reihen im Ergebnisfeld erscheinen.
Zuletzt eingeführte primäre Schlüsselkennnummer	`mysql_insert_id()`	Kein Äquivalent in PHP-Funktionen

Tab. 6.1: APIs für den Zugriff auf MySQL und Oracle

Für gewöhnlich werden Sie Ihre Datenbank nicht wöchentlich ändern, so dass dies möglicherweise für Sie nicht von so großer Bedeutung ist. Selbst mit PHPLib ist Portierbarkeit zwischen Datenbanken nur ein Gedankenspiel, solange Sie sie nicht von Beginn der Projektplanung an einbeziehen. In der Praxis zeigt es sich, dass das Problem nicht die Portierung der API ist, sondern die datenbankspezifischen Funktionen. Einen wirklich portierbaren Datenbankcode können Sie nur erzeugen, wenn Sie akzeptieren, dass Sie keine spezifischen Funktionen eines RDBMS verwenden. Dies führt jedoch dazu, dass Sie die Funktion in Ihrem Programmcode neu erzeugen müssen, was die Anwendungen möglicherweise schwerer zu pflegen und langsamer macht.

Wenn Ihr Ziel ist, portierbaren Code zu programmieren, müssen Sie die speziellen Eigenschaften der zugrunde liegenden Datenbank umgehen. PHPLib kann diese Aufgabe etwas vereinfachen. Es hat beispielsweise eine integrierte Sequenzverarbeitung und einfache Tabellensperrmechanismen, die datenbankunabhängig arbeiten. Vor kurzem haben PHPLib-Entwickler die Klasse `Query` entwickelt, welche einfache Abfragen (Einfügungen, Aktualisierungen, `where`-Anweisungen sowie einige andere, die etwa 80% der normalen Datenbankverwendung ausmachen) abstrahiert, um diese Abfragen datenbankunabhängig zu gestalten. Bisher funktioniert diese Klasse nur für MySQL und Oracle 7 (und höher).

Die Datenbank-Abstrahierungsschicht von PHPLib hat noch zwei weitere Funktionen, die mindestens genauso wichtig sind wie Portierbarkeit, und sich in ihren täglichen Anwendungen wirklich zeitsparend auswirken. Diese Funktionen werden wir in den folgenden Abschnitten beschreiben.

6.2.2 Fehlersuchmodus

Die Klasse DB_Sql hat einen Fehlersuchmodus, in welchem Sie sehen können, welche Abfragen an die Datenbank geschickt wurden. Um die Fehlersuche einzuschalten, müssen Sie lediglich eine Instanz der Klasse erzeugen und in Ihrem Programm $db->Debug = true einfügen. In unserem Programm verwenden wir in der Regel ein Konstante DEBUG, die wir der Klasse DB_Sql auf etwa diese Weise zuordnen:

```
define("DEBUG", true);
$db->Debug = DEBUG;
```

Sobald die Fehlersuche eingeschaltet ist, gibt die Klasse DB_Sql einige Werte aus den Funktionsaufrufen sowie zusätzliche Informationen aus. Dies kann sehr hilfreich sein, wenn wir einen Fehler suchen oder überprüfen möchten, ob die SQL-Abfragen korrekt sind, die Ihr Skript generiert.

6.2.3 Fehlerbehandlung

PHPLib kümmert sich um alle Fehler, die im Zusammenhang mit datenbankbezogenen Funktionen auftreten. Ein positiver Nebeneffekt ist, dass der Programmcode, den Sie mit DB_Sql erstellen, kompakter ist, weil Sie sich nicht selbst um die Fehlerbehandlung kümmern müssen.

In der Standardeinstellung wird das Skript bei jedem Fehler unterbrochen. Dieses Verhalten können Sie steuern, indem Sie die Klassenvariable $Halt_On_Error ändern, die standardmäßig auf yes gesetzt ist. Wenn Sie dies auf report setzen, gibt die Bibliothek die Fehlermeldung aus, ohne das Skript zu verlassen. Sobald die Variable auf no gesetzt ist, ignoriert die Bibliothek alle Fehler. Dies könnte zu unerwünschten Nebeneffekten, wie beispielsweise inkonsistente Daten, führen, wenn eine fehlende Datenbankabfrage ignoriert wird. Seien Sie also vorsichtig mit dieser Option. In Produktionsanwendungen sollten Fehlermeldungen informativ sein und auf die übliche und website-typischen Weise dargestellt werden. Sie können Ihre Fehlermeldungen auch selbst formatieren, indem Sie DB_Sql erweitern, indem Sie eine neue Klasse generieren, um die Funktion haltmsg() zu überschreiben. Da diese Funktion die Ausgabe aller Fehlermeldungen festlegt, können Sie Meldungen leicht auf folgende Weise verändern:

```
class test_db extends DB_Sql
{
    function haltmsg($msg)
    {
        print("Database Error: $msg<br>");
        printf("MySQL said: %s<br>", $this->Error);
    }
}
```

Die Funktion haltmsg() ist jedoch nur für die Ausgabe der Fehlermeldung zuständig. Die Unterbrechung des Skripts und Löschung des Bildschirms nach der Fehlermeldung obliegt weiterhin PHPLib. Die Funktion wird nur aufgerufen, wenn $Halt_On_Error auf yes oder report gesetzt ist. Die Variable $msg, die als Argument an haltmsg() übergeben wird, enthält eine ausführliche Beschreibung des gefundenen Fehlers. Eine von der Datenbank erzeugte Meldung erhalten Sie außerdem über $this->Error und $this->Errno.

6.2.4 DB_Sql-Beispiel

Die Verwendung der Klasse Db_Sql lässt sich am besten anhand eines kurzen Beispiels erläutern. Der Quellcode aus Listing 6.1 stellte eine Verbindung zu einer Datenbank her und zeigt den gesamten Inhalt einer einzelnen Tabelle. Er verwendet die in local.inc definierte Klasse Example_Db, welche Db_Sql erweitert, und zeigt beispielhaft, wie Sie Ihre eigenen, auf Sie angepassten, Klassen erzeugen können. Der Einfachheit halber verwenden wir in den folgenden Programmcodebeispielen die Beispielcodes aus der PHPLib-Distribution.

```
// Instantiate Example_DB class
$db = new Example_DB;

// Connect to RDBMS
$db->connect("test", "localhost", "root", "");

// Create SQL statement
$sql = "SELECT * FROM test";

// Execute query
$db->query($sql);

// Loop through result set
while($db->next_record())
{
    // Loop through the $db->Records hash
    while(list($key, $value) = each($db->Record))
    {
        // Print only non-numeric indexes
        print(is_string($key) ? "<b>$key</b>: $value<br>": "");
    }

    print("<p>");
}
```

Listing 6.1: Erstes einfaches Beispiel für die Verwendung von DB_Sql

Die erste Zeile erzeugt eine neue Instanz der Klasse DB_Sql. Standardmäßig wird diese Klasse in der Datei db_mysql.inc definiert (die in prepend.php3 geladen wird) und verwendet als Datenbankprozessor MySQL.

Als Nächstes wird eine Verbindung zur Datenbank hergestellt. Natürlich müssen Sie die Werte auf Ihre Situation anpassen. In diesem Beispiel stellen wir eine Verbindung zur Datenbank test auf localhost her, wobei wir den Benutzernamen root und kein Passwort verwenden.

Sie können diese Parameter auch direkt über die entsprechenden Klassenvariablen setzen:

```
$db = new Example_Db;
$db->Database = 'test';
$db->Host     = 'localhost';
$db->User     = 'root';
$db->Password = '';
```

Beim Aufruf von $db->query() stellt PHPLib fest, dass bisher noch keine Verbindung hergestellt wurde, und öffnet automatisch eine, wobei es die zuvor in den Klassenvariablen definierten Werte verwendet.

Unser Beispiel wird mit $db->query() fortgesetzt. Dieser Aufruf behandelt alle Aspekte, die zum Abschicken einer Abfrage an die Datenbank erforderlich sind. Er stellt eine Verbindung zur ausgewählten Datenbank her (wenn dies nicht bereits erfolgt ist) und behandelt auftretende Fehler. Wenn wir $db->Debug auf true gesetzt haben, gibt diese Funktion die SQL-Abfrage zurück, bevor sie diese an die Datenbank schickt.

Anschließend wird $db->next_record() in einer while-Schleife aufgerufen. Die Funktion erhält die nächste Zeile aus einer Reihe von Ergebnissen und speichert sie im Array $db->Record. Wenn keine weitere Zeile im Ergebnis vorhanden ist, gibt die Funktion false zurück und beendet die Schleife.

Die zweite Schleife durchsucht das Array $db->Record und gibt die Feldnamen mit ihren entsprechenden Inhalten zurück. Da dieses Array den Inhalt sowohl mit einem numerischen Index (ähnlich wie bei Arrays, die standardmäßig mit mysql_fetch_array() verwendet werden) als auch den Feldnamen als Schlüssel enthalten, stellen Sie sicher, dass nur die Array-Einträge zurückgegeben werden, bei denen der Index ein Feldname ist.

Vergleichen Sie das PHPLib-Beispiel aus Listing 6.1 mit dem Beispiel der herkömmlichen Programmierung in Listing 6.2. Sie sind in etwa gleich lang, aber das PHPLib-Beispiel besitzt alle Vorteile, die wir zuvor erläutert haben. Durch Ändern einer Datei können Sie die zugrunde liegende Datenbankschicht ändern. Auch die Fehlerbehandlung ist sehr viel leistungsfähiger als im anderen Beispiel. Durch Ändern einer Klassenvariablen weisen Sie PHPLib an, das Programm bei Fehlern zu unterbrechen, eine Fehlermeldung auszugeben

oder sie zu ignorieren, während das herkömmliche Beispiel beim Auftreten eines Fehlers einfach abbricht. Nicht zuletzt haben Sie eine vollständige Unterstützung für die Fehlersuche.

```
// Connect to RDBMS
$link = mysql_connect('localhost', 'root', '') or die(mysql_error());

// Select database
$db = mysql_select_db('test') or die(mysql_error());

// Create SQL statement
$sql = "SELECT * FROM test";

// Execute query
$res = mysql_query($sql) or die(mysql_error());

// Loop through result set
while($row = mysql_fetch_array($res))
{
    // Loop through the $db->Records hash
    while(list($key, $value) = each($row))
    {
        // Print only non-numeric indexes
        print(is_string($key) ? "<b>$key</b>: $value<br>": "");
    }

    print("<p>");
}
```

Listing 6.2: Beispiel einer herkömmlichen Programmierung

Achten Sie auf eine potentielle Falle: Sie sollten sich pro Anwendung auf eine Datenbank beschränken. PHP hat Probleme, in einem Skript den Zugriff auf unterschiedliche Datenbänke zu gewähren, insbesondere mit MySQL. PHP geht davon aus, dass es Verbindungen, die bereits hergestellt wurden, stillschweigend wiederverwenden kann, indem es denselben Benutzernamen und dasselbe Passwort verwendet. Betrachten Sie dazu folgendes Beispiel:

```
$res_one = mysql_connect("localhost", "root", "") or die(mysql_error());
$res_two = mysql_connect("localhost", "root", "") or die(mysql_error());
```

Sie würden erwarten, dass es hier zwei verschiedene Verbindungsbezeichner gibt – oder? Die gibt es jedoch nicht, da PHP die offene Verbindung im zweiten Aufruf von `mysql_connect()` wiederverwendet. Bei der Ausgabe der Verbindungsbezeichner `$res_one` und `$res_two` würden für beide Variablen derselbe Quellbezeichner angegeben. Die Auswirkung dieses Verhaltens ist, dass die Verwendung von `mysql_select_db()` für eine Verbindung auch den Kontext der anderen Verbindung ändert. Dies gilt auch für die Objekte in PHPLib:

Wenn Sie eine Datenbank für DB_Sql verwenden und eine andere Datenbank für die Sitzungsdaten, wird dies zu Problemen führen. Leider gibt es für dieses Problem bisher noch keine Lösung.

6.2.5 Sessions

PHPLib besitzt zumindest äquivalente Funktionen zu PHPs integraler Sitzungsverwaltungsbibliothek. Häufig sind sogar die Namen der Funktionen identisch. Sie ähneln zwar in vielen Aspekten der internen Sitzungsverwaltung in PHP, trotzdem ist PHPLib nicht mit ihr identisch. Eine schöne zusätzliche Funktion ist der automatische Rückfallmodus, auf den wir im nächsten Abschnitt eingehen.

6.2.6 Automatischer Rückfallmodus

Standardmäßig arbeitet die Sitzungsverwaltung mit Cookies. Wie wir im Kapitel 4 im Abschnitt »HTTP und Sessions« betont haben, sollte dies die bevorzugte Methode sein (sofern vom Client unterstützt), da es das einfachste Verfahren ist, um die Session-ID zu verbreiten. Statt dessen können Sie jedoch auch die GET / POST-Methode einsetzen, indem Sie die Variable `$mode` ändern. Die Variable `$mode` definiert, welches Verfahren primär zur Verteilung der Session-ID eingesetzt werden soll. Es kann entweder `cookie` oder `get` sein, wobei der Standardwert `cookie` ist.

PHPLib bietet zur Laufzeit einen automatischen Rückfallmodus. Wenn die Variable `$fallback_mode` auf `get` gesetzt ist, wird der GET / POST-Modus verwendet, sofern der bevorzugte Modus, der in der Variablen `$mode` angegeben ist (in der Regel `cookie`) vom Client nicht unterstützt wird. Die sinnvollste Konstellation ist, `$mode` auf `cookie` zu setzen und `$fallback_mode` auf `get`. Dementsprechend wird PHPLib versuchen, Cookies zu verwenden; wenn diese nicht unterstützt werden, »fällt es zurück« auf den GET / POST-Modus. Im Detail überprüft PHPLib die Cookie-Unterstützung folgendermaßen:

1. Bei der ersten Abfrage einer von PHPLib gespeisten Seite versucht PHPLib den Namen des Session-Cookies zu setzen, der nach der Klasseninstanz `Session` benannt ist.
2. Anschließend leitet es den Benutzer auf die Seite mit der Session-ID, die als Abfragestring angehängt ist, wobei es den folgenden Programmcode verwendet:

   ```
   header("Location: ". $PROTOCOL. "://".$HTTP_HOST.$this->self_url());
   ```

3. PHPLib prüft, ob sich die Session-ID im Array `$HTTP_COOKIE_VARS` befindet. Wenn ja, bleibt die Sitzung im Modus `cookie`. Wird die Session-ID nicht gefunden, unterstützt der Client keine Cookies und die Sitzung wird in den `GET`-Modus umgeschaltet.

6.2.7 Seiten-Caching

Mit der Klasse `Session` können Sie auch steuern, wie Seiten im Cache gepuffert werden. Die Variable `$allow_cache` kann auf `no`, `private` oder `public` gesetzt werden. Der Standardwert hängt von der PHPLib-Version ab; bei der Version 7.2 ist die Variable standardmäßig auf `private` gesetzt, bei höheren Versionen auf `no`. Der Seiten-Caching-Mechanismus ähnelt jenem, der mit den ursprünglichen Session-Funktionen von PHP 4.0 bereitgestellt wird.

6.2.8 Serialisierung

In PHP 3.0 konnten Sie Objekte noch nicht einfach serialisieren. Die Funktion `serialize()` bewahrte die Klassenmethoden nicht korrekt, und es gab keine Möglichkeit dies manuell zu tun. Der PHP 3.0-Unterstützung für Objekte fehlte eine wichtige Funktion: Introspektion. Es gab keine Möglichkeit, den Namen einer Klasse oder den Namen seiner Elternklasse zu erfahren. Daher musste PHPLib einen Umweg nehmen: Es benötigte Klassen, die zwei zusätzliche Werte hatten, nämlich `$classname` und `$persistent_slots`. Diese gaben den Namen der Klasse bzw. der Klassenvariablen für die Serialisierung an. Mithilfe des Klassennamens konnte PHPLib einen PHP-Code generieren, der eine Instanz der Klasse (`$class = new class;`) erzeugt und diese im Datenarchiv speichert. Sobald die Session-Daten wieder aufgerufen wurden, wurde der Code mit `eval()` ausgeführt. Erinnern Sie sich an das selbstmodifizierende Zählerbeispiel aus Kapitel 2 »Erweiterte Syntax«? PHPLib verwendet dieselben Konzepte.

Hinweis: Bei PHP 4.0 ist dieser Umweg nicht mehr erforderlich. PHP 4.0 besitzt Funktionen, wie `get_class()` und `get_parent_class()`, die eine bessere Introspektion der Klassen ermöglichen. `serialize()` funktioniert inzwischen für Objekte transparent.

6.2.9 Session-Beispiel

Für Ihre tägliche Arbeit ist die Verwendung von PHPLib-Session-Objekten genau so einfach wie die Verwendung der PHP 4.0-Session-Bibliothek. Das Beispiel in Listing 6.3 zeigt dies sehr deutlich. Es tut dasselbe wie das Beispiel in Kapitel 4:

```
// Create a new instance manually
$sess = new Example_Session;

// Start the session
$sess->start();

// Register our session variable
$sess->register("counter");
```

```
// Init the counter
if(!isset($counter))
{
     $counter = 0;
}

// Output session ID and counter
printf("Our session ID is: %s<br>", $sess->id);
print("The counter value is: $counter");

// Increment the counter
$counter++;

// Save the session state
$sess->freeze();
```

Listing 6.3: Ein einfaches Beispiel für die Verwendung der PHPLib-Klasse Session

Der einzige signifikante Unterschied zwischen diesem Beispiel und dem PHP 4.0-Beispiel besteht darin, dass PHPLib einen objektorientierten Ansatz verfolgt.

Genau wie die PHP 4.0-Session-Bibliothek, verwendet PHPLib Speichermodule (die sogenannten ***Container*** laut PHPLib-Terminologie), um Session-Daten zu speichern. Alle Container-Klassen beginnen mit dem Präfix `CT_`. Üblicherweise werden Session-Daten in einer SQL-Datenbank gespeichert, aber PHPLib kennt auch andere Containerklassen.

In PHPLib 7.2 sind folgender Container-Klassen verfügbar:

- Der Standard-Container ist `CT_Sql`.. Er speichert die Session-Daten in einer Datenbank. Er besitzt folgende Klassenvariablen:

Name	Beschreibung
`$database_class`	Name der Klasse `DB_Sql`, die für den Verbindungsaufbau mit der Datenbank verwendet werden sollte.
`$database_table`	Name der Tabelle, die zur Speicherung der Session-Daten verwendet wird.
`$encoding_mode`	Diese Variable steuert die Art, wie Session-Daten gespeichert werden. Sie kann zwei Werte annehmen : `base64` oder `slashes`. Normalerweise sollten Sie den Standardwert (`base64`) nicht ändern. Auf diese Weise werden Session-Daten mit Base64 verschlüsselt, bevor sie in der Datenbank gespeichert werden. Möglicherweise möchten Sie aber die alternative Methode `slashes` verwenden, um Session-Daten als Klartext in der Tabelle abzulegen und die Fehlersuche zu erleichtern.

- In Bezug auf Funktionen ist CT_Split_Sql identisch mit CT_Sql. Verwenden Sie diese, wenn die zugrunde liegende Datenbank nicht genügend Daten in einem Feld für die Session-Daten speichern kann, insbesondere, wenn die Datenbank Probleme mit großen binären Objekten (BLOBs) hat. CT_Split_Sql ist nicht mit den Tabellen von CT_Sql kompatibel.
- Mit der Variablen $split_length können Sie die Länge festlegen, an der die Klasse die Session-Daten trennt. Der Standardwert hierfür ist 4096 (4KB).
- CT_Shm speichert Session-Daten im gemeinsam genutzten Speicher, dem sogenannten Shared Memory. Um diesen Container zu verwenden, müssen Sie PHP mit Unterstützung für den gemeinsamen Speicher kompilieren. Dieser Container ist schneller, da er Session-Daten direkt im Speicher verfügbar ablegt. Der Nachteil ist, dass alle Session-Daten verloren gehen, wenn Sie Ihren Server aus irgendeinem Grund neu starten müssen. Außerdem ist die Anzahl der gleichzeitig offenen Sessions durch die Speicherkapazität begrenzt. Jede Session beansprucht einen gewissen Speicherplatz (die Höhe hängt von der Menge und Größe der Sitzungsvariablen ab); sobald der gesamte verfügbare Speicher belegt ist, können keine weiteren Sitzungen gestartet werden.
- Folgende Klassenvariablen unterscheiden sich von jenen aus CT_Sql:

Variable	Beschreibung
$max_sessions	Höchstzulässige Anzahl an gleichzeitig aktiven Sessions. Der aktuelle Standardwert ist 500.
$shm_key	Eindeutiger Schlüssel für das Segment des Shared Memory, das verwendet werden sollte. Es ist wichtig, dass dieser Schlüssel für jede Anwendung eindeutig ist.
$shm_size	Größe des Segments des Shared Memory in Bytes. Die Größe lässt sich ungefähr nach der Formel shm_size = max_sessions * session_size berechnen, wobei die Session-Größe durchschnittlich 600 Byte groß sein konnen. Der Standardwert ist 64.000 (64KB).

- CT_Dbm verwendet zur Speicherung der Session-Daten die Datei UNIX DBM. Dieser Datenbanktyp speichert Daten als Schlüssel / Wert-Paare in regulären Dateien auf dem System. Als einzige Variable sollten Sie $dbm_file setzen; diese gibt den Dateinamen Ihrer DBM-Datei an. Die Datei muss die entsprechenden Rechte haben, und der Server braucht einen Schreibzugriff auf diese.
- CT_Ldap speichert Session-Daten auf einem LDAP(Lightweight Directory Access Protocol)-Server. Um diesen Container zu verwenden, müssen Sie PHP mit LDAP-Unterstützung kompilieren. Die Klasse CT_Ldap besitzt folgende Eigenschaften:

Variable	Beschreibung
$ldap_host, $ldap_port	Hostname und Port-Nummer des LDAP-Servers
$rootdn, $rootpw	Vom Stamm erkannter Namen und Passwort des LDAP-Servers, die zur Verbindung mit dem Server verwendet werden
$basedn	Unterhalb des angegebenen Namens sollten die Session-Daten gespeichert werden
$objclass	Name der Objektklasse (vergleichbar mit dem Namen einer SQL-Tabelle)

Bei einem Blick auf `local.inc` sehen Sie eine Reihe von Klassendefinitionen, von denen drei unser Beispiel betreffen:

```
class DB_Example extends DB_Sql {
  var $Host     = "localhost";
  var $Database = "phplib";
  var $User     = "tobias";
  var $Password = "justdoit";
}

class Example_CT_Sql extends CT_Sql {
  var $database_class = "DB_Example";      ## Which database to connect...
  var $database_table = "active_sessions"; ## and find our session data in
                                              ➥this
}

class Example_Session extends Session {
  var $classname = "Example_Session";

  var $cookiename     = "";              ## defaults to classname
  var $magic          = "Hocuspocus";    ## ID seed
  var $mode           = "cookie";        ## We propagate session IDs with
                                            ➥cookies
  var $fallback_mode  = "get";
  var $lifetime       = 0;               ## 0 = do session cookies, else
                                            ➥minutes  until the session
                                            ➥expires
  var $that_class     = "Example_CT_Sql"; ## name of data storage container
  var $gc_probability = 5;
}
```

Wie Sie sehen, haben die Klassen eine Beziehung. In der Klasse `Example_Session`, ist die Varibale `$that_class` auf den Namen der Klasse `Example_CT_Sql` gesetzt, und in `Example_CT_Sql` zeigt die Variable `$database_class` auf die Klasse `DB_Sql`. Abbildung 6.1 zeigt diese Beziehung im Detail.

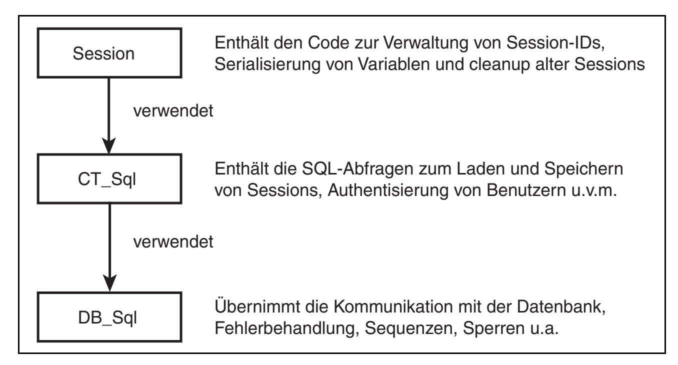

Abbildung 6.1: Beziehungsmodell der Klassen DB_Sql, CT_Sql, und Session

Unser Beispiel verwendete keine Basisklassen, sondern Erweiterungen davon, und zwar so, wie sie in `local.inc` definiert sind. Abbildung 6.2 zeigt die Abhängigkeiten.

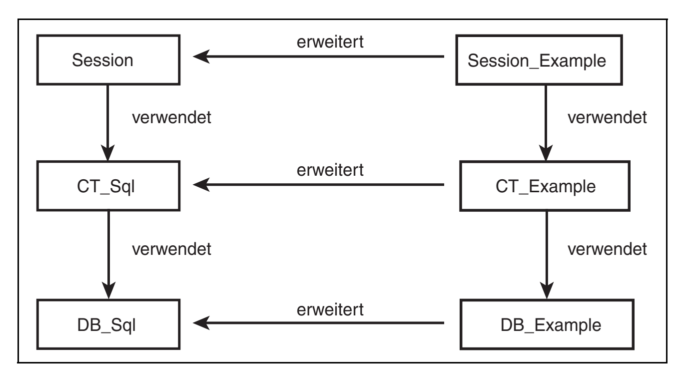

Abbildung 6.2: Im Beispiel vorhandene Abhängigkeiten und Beziehungen

6.2.10 Abkürzungen I: page_open()

Angenommen, Sie wollten eine größere Anwendung entwickeln und dabei Session-Verwaltung, Datenbankabstrahierung und PHPLib-Funktionen für die Authentisierung und Verwaltung von Zugriffsrechten verwenden. Dafür müssen Sie Instanzen der Session-, Authentisierungs- und Berechtigungsobjekte erzeugen. Ihre Datei `local.inc` sieht in etwa folgendermaßen aus:

```
$sess = new Session_Example;
$sess->start();

$auth = new Auth_Example;
$auth->start();
```

```
$perm = new Perm_Example;
$user = new User_Example;
$user->start();
```

Da die Klassen voneinander abhängen, müssen Sie diese in der richtigen Reihenfolge initialisieren. Sie können keine Instanz von User_Example erzeugen, bevor Sie nicht eine Instanz der Session- und der Authentisierungsinstanz generiert haben. Das ist aber noch nicht alles. Wie das Ende Ihres Programms aussieht, hängt davon ab, welche Klassen Sie zuvor geöffnet haben. Es ist wichtig, in welcher Reihenfolge die Cleanup-Routinen der Klassen aufgerufen werden.

PHPLib hilft Ihnen hier mit den Funktionen page_open() und page_close(). In der Dokumentation werden sie als Seitenverwaltungsfunktionen bezeichnet. Tatsächlich können sie den gesamten Verbindungsauf- und -abbau von PHPLib bewerkstelligen. Mithilfe dieser beiden Funktionen können wir unser Beispiel beträchtlich verkürzen.

```
page_open(array("sess" => "Session_Example",
                "auth" => "Auth_Example",
                "perm" => "Perm_Example"));

[...]

page_close();
```

Die Funktion page_open() in unserem Beispiel erzeugt Instanzen von Session_Example, Auth_Example und Perm_Example, mit Namen $sess, $auth bzw. $perm. Diese Instanzen können Sie anschließend direkt benutzen, z.B. mit $sess->register().

Hinweis: Die Funktion page_open() muss vor der Ausgabe aufgerufen sein, da sie einen Cookie und weitere HTTP-Header setzt.

6.2.11 Abkürzungen II: purl(), url() und pself()

Wenn Ihre Anwendung als Primärmodus für die Verteilung der Session-ID cookie verwendet und get als Fallback-Modus, müssen Sie Ihre gesamten Links mit der Session-ID kennzeichnen. Um andere Methoden für die Verbreitung der Session-ID zu verwenden, die wir in Kapitel 4 vorgestellt haben, müssen Sie PHPLib um einen zusätzlichen Wert für $mode erweitern.

PHPLib vereinfacht sogar die manuelle Neuschreibung von URLs. Es stellt die Funktion url() bereit, welche die Session-ID an den Link anhängt, sobald es in den get-Modus wechselt.

```
$link = $sess->url("script.php3");
```

Wenn Ihre Session derzeit im `get`-Modus ist, sähe die dabei erzeugte Variable `$link` etwa folgendermaßen aus:

```
script.php3?Example_Session=2e4c3670ce9a143fee398aec282f960c
```

Diese Funktion handhabt sogar Abfragezeichenketten korrekt. Selbst wenn Sie eine URL aufrufen, die einen Parameter, beispielsweise `script.php3?foo=bar` enthält, entstehen keine Probleme.

Als Kurzform können Sie für die Ausgabe des entstandenen Links die PHPLib-Funktion `purl()` verwenden, die genauso funktioniert wie `url()`, aber auch die generierte URL ausgibt. `self_url()` und `pself_url()` erzeugen entsprechend (mit letzterer Funktion) einen mit Tags gekennzeichneten Link auf die aktuelle Datei, und geben diesen aus.

6.3 Authentisierung

Im Abschnitt »Authentisierung« in Kapitel 4 haben wir erwähnt, dass die HTTP-Authentisierung einige Nachteile hat und dass Sie diese mit der PHP-basierten Authentisierung umgehen können. In den folgenden Abschnitten gehen wir hierauf ausführlich ein.

6.3.1 Vorteile der PHP-Genehmigung

Dieser Abschnitt setzt dort an, wo wir den Abschnitt »Authentisierung« des Kapitel 4 verlassen haben. Dort erläuterten wir, dass die HTTP-Authentisierung einige Nachteile hat und dass wir diese mit der PHP-basierten Authentisierung umgehen können. PHPLib hat Klassen zur besseren Handhabung der Benutzer-Authentisierung und Verwaltung von Zugriffsrechten.

PHPLib erteilt Authentisierungen für Sessions; dementsprechend hängt es von der Klasse `Session` ab. Auf denjenigen Seiten, auf denen Sie eine Authentisierung benötigen, sollte die folgende Anweisung `page_open()` aufgerufen werden, um eine Instanz einer Session und einer Authentisierung zu erzeugen.

```
page_open(array("sess" => "Session_Example", "auth" => "Auth_Example"));
```

Sessions als Basis für die Authentisierung heranzuziehen, bringt eine Reihe von Vorteilen:

- Der Benutzername und das Authentisierungselement werden nur einmal, und zwar bei der Anmeldung, verschickt. Sobald er authentisiert ist, speichert der Server die Authentisierungsdaten in der Session und überträgt den Benutzernamen oder das Authentisierungselement kein weiteres mal. Hier unterscheidet sich PHP von HTTP, bei der der Benutzername und das Passwort in den HTTP-Headern bei allen Abfragen übertragen werden.

Dies bedeutet jedoch auch, dass Sie die Authentisierung verlieren, sobald die Session verloren geht.

- Der Authentisierungsprozess auf dem Server kann kompliziert sein. Er kann über eine Datenbank oder einen beliebigen anderen Mechanismus ausgeführt werden. Die Authentisierung wird über eine nicht definierte Funktion der Klasse `Auth` durchgeführt (`auth_validatelogin()`), und die Implementierung obliegt Ihnen.
- Die Authentisierung ist nicht auf das Verzeichnis beschränkt, sondern kann für einzelne Dateien der Anwendung unterschiedlich sein; Sie können sogar in einem Skript verschiedene Authentisierungsebenen realisieren. Sie können Teile des Skripts vor Benutzern verbergen, die nicht berechtigt sind, auf diese Teile zuzugreifen.
- Benutzer, die dem System nicht bekannt sind, können sich vor der Anmeldung registrieren lassen. Ein Registrierungsformular wird eingeblendet, und PHPLib erzeugt automatisch einen Standardeintrag in der Benutzerdatenbank.
- Die Authentisierung über PHPLib funtioniert sogar mit die CGI-Version von PHP.
- Sie können Benutzer sauber abmelden. Dies bedeutet, der Benutzer erhält die Möglichkeit, die aktuelle Session zu beenden (über die entsprechende Schaltfläche).
- Benutzer können nach einer gewissen Leerlaufzeit automatisch abgemeldet werden. Damit bekommt Ihre Anwendung eine zusätzliche Sicherheit, da diese Funktion das Kapern von Nutzungen nach längerer Leerlaufzeit verhindert.

6.3.2 Auth-Beispiel

Beginnen wir mit einen einfachen Beispiel. Das Beispiel in Listing 6.4 zeigt den Standardanmeldebildschirm von PHPLib, wenn Sie diesen das erste Mal aufrufen. Die Anmeldung erfolgt über ein Standardbenutzername / Passwort-Paar von `kris/test`. Sobald Sie authentisiert wurden, sehen Sie Ihre Session-ID, Ihren Benutzernamen und Ihre Zugriffsberechtigungen.

```
page_open(array("sess" => "Example_Session", "auth" => "Example_Auth"));

printf("Your session id: %s<p>\n", $sess->id);
printf("Your user ID: %s<br>\n", $auth->auth["uid"]);
printf("Your user name: %s<br>\n", $auth->auth["uname"]);
printf("You permissions: %s<br>\n", $auth->auth["perm"]);

page_close()
```

Listing 6.4: Ein einfaches Beispiel für die Verwendung von Auth.

Alle Seiten, welche die PHPLib-Authentisierung verwenden, haben diese allgemeine Struktur. Zunächst wird page_open() aufgerufen. Der Rest des Skripts wird erst ausgeführt, wenn der Benutzer angemeldet und authentisiert ist. Sie können sicher davon ausgehen, dass kein Benutzer irgendwelche Informationen unterhalb von page_open() sieht, ohne dass er angemeldet ist. Bei einem einzeiligen Aufruf von page_open() haben Sie die vollständige Benutzerauthentisierung in Ihrem Skript hinzugefügt. Nachdem Sie die Klassen definiert haben, die Sie in der Anwendung benutzen möchten, ist PHP tatsächlich so leicht zu verwenden. Unsere Beispiele haben bisher die Beispielklassen verwendet, welche die PHPLib-Distribution bereitstellt. In Ihrer Arbeit möchten Sie möglicherweise Ihre eigenen Klassen erzeugen, die von den Basisklassen abgeleitet wurden, um sie stärker an Ihre eigenen Bedürfnisse anzupassen. Dafür ist es notwendig, dass Sie die interne Funktionsweise von PHPLib besser verstehen.

6.3.3 Auth-Interna

Wenn wir davon ausgehen, dass Sie als Datenbank MySQL verwenden, wird für Ihre Benutzertabelle folgendes Schema verwendet:

```
CREATE TABLE auth_user (
   user_id varchar(32) NOT NULL,
   username varchar(32) NOT NULL,
   password varchar(32) NOT NULL,
   perms varchar(255),
   PRIMARY KEY (user_id),
   UNIQUE k_username (username)
);
```

Der primäre Schlüssel ist user_id, da PHPLib intern nicht mit dem Benutzername / Passwort-Paar des Benutzers arbeitet, sondern mit dieser ID. Diese ID (die bei PHPLib uid heißt) ist eine eindeutige Zeichenfolge, vergleichbar mit der Session-ID, die aus einer Kombination von uniqid() und md5() erzeugt wird.

```
$uid = md5(uniqid($hash_secret));
```

Warum verwendet PHPLib nicht einfach einen primären Schlüssel, der sich aus den Feldern username und password zusammensetzt? Auf diese Weise könnte man sich das zusätzliche Feld user_id sparen. Die Antwort ist einfach: PHPLib soll für einen beliebigen Authentisierungsprozess einsetzbar sein und den Verbindungsaufbau so einfach wie möglich halten. Wenn Sie für jeden Benutzer einen separaten eindeutigen Bezeichner fester Länge nehmen, können Sie einfach zusätzliche Tabellen erstellen, die über eine relationale Datenbank mit der auth_user-Tabelle verknüpft werden.

In unserem letzten Beispiel funktionierte der Trick, weil wir einfach die Standardimplementierung der Klasse Auth verwendet haben, die in der PHPLib-Distribution in Example_Default_Auth mitgeliefert wird. In den meisten Fällen müssen Sie jedoch eine eigene Klasse definieren, indem Sie die Auth-Basisklasse erweitern. In der reinen Form ist Auth nicht zu gebrauchen, da ihm zwei Funktionen fehlen, die wir für die Authentisierung benötigen.. Auth weiß weder, wie Ihr Anmeldebildschirm aussehen soll, noch, wie Sie den Authentisierungsprozess gestalten möchten. Da Auth dies für Sie nicht übernimmt, müssen Sie diese Funktionen in Ihren abgeleiteten Klassen selbst definieren. Listing 6.5 zeigt ein Beispiel für eine solche abgeleitete Klasse, die jener ähnelt, die sich in local.inc als Beispielklasse befindet.

```
require("EasyTemplate.inc.php3");
class My_Auth extends Auth
{
    var $classname = "My_Auth";
    var $database_class = "DB_Example";
    var $database_table = "auth_user";

    function auth_loginform()
    {
        // Create template instance
        $tpl = new EasyTemplate("loginform.inc.html");

        // Is the username already set? If yes, it means that the
        // first authentication try failed.
        if(isset($this->auth["uname"]))
        {
            $tpl->assign("USERNAME", $this->auth["uname"]);
            $tpl->assign("MESSAGE", "Either your username or your password
            ➥are invalid.<br> Please try again!");
        }
        else
        {
            $tpl->assign("USERNAME", "");
            $tpl->assign("MESSAGE", "Please identify yourself with a
            ➥username and a password:");
        }

        // Assign action to form, which points to ourselves
        $tpl->assign("ACTION", $this->url());

        // Output the parsed template
        $tpl->easy_print();
    }
```

```
    function auth_validatelogin()
    {
        // Global form variables
        global $username, $password;

        // If $username is set, remember it
        if(isset($username))
        {
            $this->auth["uname"] = $username;
        }

        // Set the $uid to false by default
        $uid = false;

        // Select rows corresponding to the submitted username/password
        $query = "
                SELECT
                  *
                FROM
                  $this->database_table
                WHERE
                   username = '$username'
                   AND password = '$password'
                ";

        // Execute query
        $this->db->query($query);

        // If one row was returned, the user is authenticated
        if($this->db->num_rows() == 1)
        {
            $this->db->next_record();

            // Set up $uid and $this->auth array.
            $uid = $this->db->Record["user_id"];
            $this->auth["uid"] = $uid;
            $this->auth["uname"] = $this->db->Record["username"];
        }

        return($uid);
    }
}
```

Listing 6.5: Erweiterung der Auth-Basisklasse

Intern verwendet Auth die beiden Klassenvariablen $database_class und $database_table, um Session- und Authentisierungsdaten zu speichern. Sie haben keinen Einfluss auf den Authentisierungsvorgang. Die Anmeldung wird von den beiden Klassenfunktionen verwaltet, die Sie definieren müssen: auth_loginform() und auth_validatelogin().

Wenn ein Benutzer eine geschützte Seite anfordert, aber noch nicht angemeldet ist, ruft die Klasse Auth die Funktion auth_loginform() auf. Diese Funktion sollte einen Anmeldebildschirm erzeugen. Da sie auch aufgerufen wird, wenn die Authentisierung fehlschlägt, sollte sie auch über Mechanismen verfügen, um auf fehlgeschlagene Versuche reagieren zu können. In unserem Beispiel wird eine entsprechende Fehlermeldung angezeigt, und im Benutzerfeld des Formulars wird der übertragene Wert eingetragen.

Die zweite Funktion auth_validatelogin() bildet den Kern der Klasse. Sie führt die Authentisierung durch. Diese Funktion wird aufgerufen, nachdem der Benutzer die Authentisierungsdaten aus dem Formular übertragen hat, die von der Funktion auth_loginform() bereitgestellt wurden. Die Variablen des Formulars werden hier als globale Variablen verwendet und sollten daher in der Funktion als solche definiert sein, damit auf sie zugegriffen werden kann. Wie Sie die Authentisierung durchführen, bleibt Ihnen überlassen. Im Beispiel führen wir einen Abgleich mit der Standard-auth_user-Tabelle aus der PHPLib-Distribution durch. Statt dessen könnten Sie auch htaccess-Stildateien, einen LDAP-Server o. ä. verwenden.

Wenn die Authentisierung erfolgreich ist, muss die Funktion eine gültige Benutzerkennnummer zurückgeben und das Array $this->auth einrichten. Dieses assoziative Array muss mindestens zwei Elemente enthalten: die eindeutige Benutzerkennnummer uid und den vom Benutzer eingegebenen Benutzernamen uname.

Wenn Sie mit der Klasse Perm Berechtigungsebenen einrichten möchten (mehr dazu später), müssen Sie ein weiteres Element einrichten, nämlich $this->auth["perm"]. Dieses Element sollte eine Liste aller Berechtigungen enthalten, die der Benutzer hat, wobei die einzelnen Berechtigungen durch ein Komma ohne Leerzeichen voneinander getrennt sind, z.B. admin oder author,editor. In der Regel wird diese Liste aus demselben Speichermedium ausgelesen, aus dem Sie auch die Benutzerdaten erhalten, in unserem Beispiel die MYSQL-Datenbank.

Wenn die Authentisierung fehlschlägt, muss die Funktion false zurückgeben, und die Funktion auth_loginform() wird erneut aufgerufen. Haben Sie bemerkt, dass wir $this->auth["uname"] eingerichtet haben, ungeachtet dessen, ob die Authentisierung erfolgreich war? Das Array $this->auth ist eine Session-Variable und bleibt daher über mehrere Anmeldeversuche hin erhalten. In der Funktion auth_loginform() überprüfen wir auch, ob der Benutzernamen bereits angegeben wurde, und füllen das Anmeldeformular dementsprechend aus.

Nachdem sich der Benutzer in Ihrer Anwendung korrekt angemeldet hat, wissen Sie genau, mit wem Sie es zu tun haben. Zur Verwaltung der unterschiedlichen Berechtigungsebenen, die mit den einzelnen Benutzern verknüpft sind, können Sie eine weitere PHPLib-Klasse namens Perm verwenden.

6.3.4 Verwaltung von Berechtigungsebenen

In einer typischen Anwendung haben Sie in der Regel zwei Berechtigungsebenen: Benutzer und Administratoren. Bei einigen Anwendungen brauchen Sie jedoch eine detailliertere Zugriffskontrolle. Ein Content Management System braucht beispielsweise viele Berechtigungsebenen:

- Einen Superuser, der alles im System ändern kann, um das Benutzersystem zu verändern usw.
- Editoren, die Artikel und Inhalte bearbeiten, sowie den Inhalt genehmigen können, der von den Autoren kommt
- Autoren, die Inhalte erzeugen können und sie zur Genehmigung an die Editoren weiterleiten, aber keine Inhalte genehmigen können
- Benutzer mit Nur-Lese-Zugriff

Da Sie die derzeit angemeldeten Benutzer kennen und den Benutzer anhand seiner eindeutigen Kennnummer ($uid) identifizieren können, wäre es nicht sehr schwer, Funktionen zu schreiben, mit denen Sie dem Benutzer einer Gruppe zuordnen können, und ihm entsprechend dieser Gruppe Daten anzeigen. PHPLib bietet hierfür eingebaute Funktionen.

Um die Klasse Perm zu verwenden, müssen Sie im page_open()-Aufruf ein weiteres Element hinzufügen. PHPLib bietet eine Standardimplementierung der Klasse Perm namens Example_Perm. Doch hier gilt dasselbe Prinzip wie zuvor: Um alle verfügbaren Funktionen nutzen zu können und an Ihre spezifischen Anforderungen anzupassen, sollten Sie sich eine eigene Klasse in local.inc definieren.

Listing 6.6 zeigt ein Beispiel für die Verwendung von Perm. Es enthält ein paar mehr Funktionen als notwendig, da Sie sich in diesem Beispiel abmelden und mit einem anderen Benutzernamen wieder anmelden können. Dies ermöglich uns, leichter festzustellen, welche Berechtigungsebenen in Gebrauch sind. Der von PHPLib zur Verfügung gestellte Beispielbenutzer (Benutzername kris, Passwort test) hat admin-Privilegien. Wenn Sie sich mit diesem Benutzernamen anmelden werden Sie mit der Nachricht »Welcome Admin« begrüßt. Da die PHPLib-Distribution nur diesen einen Benutzer hat, müssen Sie einen neuen Benutzer generieren, wenn Sie wissen möchten, wie die Seite für Benutzer mit Privilegien unterhalb von admin aussieht.

```
page_open(array("sess" => "Example_Session", "auth" => "Example_Auth",
➥"perm" => "Example_Perm"));

if(isset($mode) && $mode == "reload")
{
    $auth->unauth();
    print("You have been logged out.<br>");
    printf('If you want, you can <a href="%s"> login again.',
    ➥$sess->url(basename($PHP_SELF)));
}
else
{
    if($perm->have_perm("admin"))
    {
        print("<b>Welcome Admin.</b><br>");
        print('You are logged in with "admin" permissions.<br>');
    }
    else
    {
        printf('You are logged in with "%s" permissions.<br>',
        ➥$auth->auth["perm"]);
    }

    printf("Your user name: %s<br>", $auth->auth["uname"]);
    printf('<a href="%s">Log out</a>',
    ➥$sess->url(basename($PHP_SELF)."?mode=reload"));
}

page_close();
```

Listing 6.6: Verwendung der Klasse Perm.

6.3.5 Bitweise Berechnungen

Bitweise Berechnungen stiften bei unerfahrenen Programmierern häufig Verwirrung, und selbst erfahrene Entwickler haben manchmal Probleme damit. Die Darstellung von Flagwerten in Bitmustern kann trotzdem sehr nützlich sein. PHPLib verwendet diese Form für die Berechtigungsebenen. Oft wird dieses Verfahren auch dazu verwendet, Flagwerte in einem einzelnen INT-Feld einer Datenbank zu speichern. Nehmen wir als Beispiel eine Anwendung, welche die verschiedenen Zustände, beispielsweise die Hobbys des Benutzers, protokollieren muss. Anstatt für jedes Hobby ein eigenes Feld in einer Datenbank anzulegen und dieses auf true oder false zu setzen, können Sie ein einzelnes Flagfeld verwenden. Je nachdem ob der Benutzer ein bestimmtes Hobby hat oder nicht, wird das Bit für dieses Hobby gesetzt oder nicht.

In einfachen Worten sind bitweise Operationen Operationen, die ein oder mehrere Bits gleichzeitig verarbeiten. Sie wissen, dass das Binärsystem aus einer Achterfolge von Bits besteht – einer Reihe von Nullen und Einsen. Die Dezimalzahl 42 wird im binären System als 00101010 dargestellt:

```
Bit-Position: 7 6 5 4 3 2 1 0
Bit-Wert :    0 0 1 0 1 0 1 0
```

Das Bit ganz rechts außen, Bit 0, wird als niedrigstwertiges Bit bezeichnet. Entsprechend ist Bit 7 das höchstwertige Bit. Um Daten vom binären System in das dezimale System umzuwandeln und umgekehrt, können Sie in PHP `BinDec()` bzw. `DecBin()` verwenden.

Die Bits in diesen Oktetts werden von binären Operatoren aktiviert bzw. deaktiviert.

Setzen eines Bit

Zum Setzen eines Bits in einem Wert verwenden Sie das Inklusiv-ODER (`value | value`). Wenn der aktuelle Wert des Flags drei ist (also das erste und zweite Bit gesetzt sind) und Sie das dritte Bit setzen möchten (bedenken Sie, die Bitzählung beginnt bei Null), wenden Sie ODER auf die aktuellen Werte 3 und 4 (zwei hoch zwei) an:

```
$value = 3;
$value |= 4;
```

Der Wert ist jetzt sieben. Im Binärsystem sieht die Operation folgendermaßen aus:

```
  0 0 0 0 0 1 1
| 0 0 0 0 1 0 0
= 0 0 0 0 1 1 1
```

Umschalten eines Bit

Das Umschalten eines Bit – d.h. das Setzen auf Eins, wenn es auf Null ist, und umgekehrt – geschieht über den Exklusiv-ODER-Operator (XOR) (`value ^ value`). Wenn unser Wert 3 ist und wir das erste Bit umschalten wollen (das derzeit Eins ist: `0000011`), müssten wir folgende Zeile verwenden:

```
$value = 3;
$value ^= 2;
```

Das Ergebnis wäre Eins. In Binärschreibweise sieht dies folgendermaßen aus:

```
  0 0 0 0 0 1 1
^ 0 0 0 0 0 1 0
= 0 0 0 0 0 0 1
```

Löschen eines Bit

Einen Bit löschen Sie am einfachsten, indem Sie zunächst sicherstellen, dass es gesetzt ist, und es anschließend umkehren. Dazu benötigen Sie zwei Bitoperatoren: `INVERS` und `UND` (`value & ~value`). Mit folgender Zeile können Sie das erste Bit des Wertes 3 löschen:

```
$value = 3;
$value &= ~2;
```

Überprüfen eines Bit

Um zu überprüfen, ob ein Bit in einem Wert gesetzt ist, verwenden Sie den logischen UND-Operator. Er vergleicht zwei Werte. Wenn beide Bits Eins sind, gibt er Eins zurück. Um beispielsweise zu überprüfen, ob das erste Bit im Wert 3 gesetzt ist, würden Sie folgende Zeile verwenden:

```
if(2 & 3)
   // more code
```

Bitweise Verschiebung

Um Bits nach links oder rechts zu verschieben, verwenden Sie die Verschiebeoperatoren << und >>. Wenn Sie beispielsweise den Binärwert 1 um binär 1 Stelle nach links verschieben, erhalten Sie den Binärwert 10:

```
     0 0 0 0 0 0 1
   <<0 0 0 0 0 0 1
  =  0 0 0 0 0 1 0
```

Dies kann nützlich sein, um Bits zu setzen, wie wir später im Beispiel zeigen werden.

Operatorrangfolge

Es ist gar nicht schlecht, sich einmal die Rangfolge bei bitweisen Operatoren anzusehen. Bedenken Sie, dass arithmetrische Operatoren Vorrang vor bitweisen Operatoren haben. Die Anweisung `1 + 2 | 3` wird ausgewertet als `(1+ 2) | 3`. Sie sollten auch nicht vergessen, dass die bitweisen Operatoren den Vergleichoperatoren nachgestellt sind. Hüten Sie sich also davor, Anweisungen wie `if(2 & 3 != 0)` zu schreiben. Anstatt zu testen, ob das zweite Bit im Wert drei gesetzt ist (es ist), überprüft die Anweisung zunächst `3 !=0` und gibt `true` zurück, was zu `2 & 1` führt. Dadurch wird `0` zurückgegeben, was definitiv nicht das ist, was Sie wollten.

Beispiel

Kehren wir zu unserem Hobby-Beispiel zurück. Angenommen, der Benutzer kann zwischen vier verschiedenen Hobbys wählen: Lesen, Programmieren, Schreiben und Wandern. Zunächst ordnen wir jedem Hobby ein Bit zu:

- Lesen: 1
- Programmieren: 2
- Schreiben: 4
- Wandern: 8

In der Binärschreibweise werden die Werte folgendermaßen dargestellt:

```
Lesen:           0 0 0 0 0 0 1
Programmieren:   0 0 0 0 0 1 0
Schreiben:       0 0 0 0 1 0 0
Wandern:         0 0 0 1 0 0 0
```

Im Programmcode können Sie die Hobbys durch eine einfache Bitverschiebung definieren:

```
define("HOBBY_READING", 1 << 0);
define("HOBBY_PROGRAMMING", 1 << 1);
define("HOBBY_WRITING", 1 << 2);
define("HOBBY_HIKING", 1 << 3);
```

Wenn der Benutzer nun die Hobbys Schreiben und Programmieren wählt, können Sie ein Bitmuster erstellen, indem Sie die beiden Werte in ODER-Schreibweise miteinander verknüpfen:

```
$pattern = HOBBY_WRITING | HOBBY_PROGRAMMING;
```

Später können Sie überprüfen, ob der Benutzer als Hobby Schreiben gewählt hat, indem Sie das zugehörige Bit im Bitmuster überprüfen:

```
printf("Writing %s chosen.", (HOBBY_WRITING & $pattern) ? "is": "is not");
```

Um das Schreib-Bit auf Null zu setzten, verwenden Sie folgende Zeile:

```
$pattern &= ~HOBBY_WRITING;
```

Die Überprüfung der erforderlichen Berechtigungsebene erfolgt über die Funktion `$auth->have_perm()`. Die Berechtigungsebene, die Sie überprüfen möchten, übergeben Sie als Argument. In unserem Beispiel verwenden wir `$auth->have_perm("admin")`, um zu kontrollieren, ob der Benutzer das Privileg `admin` hat. Wenn der Benutzer die erforderliche Berechtigungen hat, gibt die Funktion `true` zurück, ansonsten `false`.

Jedem Benutzer werden Privilegien zugeordnet. Bei Verwendung der Funktion `Example_Auth` werden die Privilegien in der MySQL-Tabelle `auth_user` gespeichert. Sie kennen das Feld bereits:

```
perms varchar(255)
```

Eine Liste gültiger Berechtigungen wird in der Klasse definiert, die von Perm angeleitet wurde, und zwar in der Klassenvariablen $permissions. Die Klasse Example_Perm hat folgende Berechtigungen:

```
var $permissions = array(
        "user"       => 1,
        "author"     => 2,
        "editor"     => 4,
        "supervisor" => 8,
        "admin"      => 16
         );
```

Berechtigungen werden intern in Bitmaps umgewandelt und mit logischen UND und ODER berechnet. Die Werte in diesem assoziativen Array definieren für jede Berechtigungsebene ein Bitmuster. Das mag zwar ein wenig kompliziert klingen, aber die Verwendung von Bitmustern bringt für Szenarien, wie Berechtigungsebenen, einige Vorteile. Zum Beispiel werden automatisch Vorkehrungen für Ebenen getroffen, welche die Berechtigungen aus niedrigeren Ebenen erben: Eine admin-Ebene hat automatisch auch die user-Privilegien, wenn Sie die Bitmuster entsprechend konzipieren.

Bei der Standardeinstellungen werden diese Vorkehrungen jedoch nicht getroffen. Die admin-Ebene unterscheidet sich von der user-Ebene, so dass ein Benutzer, der zur admin-Gruppe gehört, nicht auf Funktionen zugreifen kann, die mit $auth->have_perm("user") gesichert wurden. Um Ihnen ein besseres Verständnis für die Funktionsweise von PHPLib zu geben, zeigen wir Ihnen, wie es Bitmuster berechnet:

- Der Zugriff auf die Funktionen erfolgt nur über die user-Ebenen. Diese Ebene hat das Bitmuster 1.
- Der Benutzer befindet sich auf der admin-Ebene, welche das Bitmuster 16 hat.
- Diese beiden werden mit einem logischen UND verknüpft, was Null ergibt (überprüfen Sie es selbst: print(16 & 1);). Das Ergebnis 0 ist nicht identisch mit der angeforderten Ebene (1), deshalb wird der Zugriff verweigert.

Die Berechnung ist eigentlich nichts anderes, als dass PHPLib überprüft, ob im Bitmuster, das als Argument von $perm->have_perms() angegeben wurde, das Berechtigungsbit des Benutzers gesetzt wurde. Diese Vorgehensweise ermöglicht komplexe Kombinationen.

Betrachten wir ein weiteres Beispiel: Angenommen, Sie haben vier Berechtigungsebenen: admin, supervisor, editor und author. Sie möchten, dass die Editoren Inhalte nicht weiterleiten dürfen (author-Ebene), die anderen Gruppen sollen jedoch die Berechtigungen aus den Ebenen unter ihnen übernehmen. Das endgültige Berechtigungssystem sieht dann folgendermaßen aus:

- `admin`: **erbt** `supervisor, editor, author`
- `supervisor`: **erbt** `editor, author`
- `editor`
- `author`

Um die Bitmuster der einzelnen Gruppen zu berechnen, beginnen Sie bei der untersten Ebene `author` mit dem Bitmuster von 1 (was bedeutet, dass das ganz rechte Bit gesetzt ist). Wenn wir möchten, dass der Editor die Rechte der Autorebene erbt, müssten wir mit dem Bitmuster für Autoren fortfahren und das nächsthöhere Bit setzten (`1 | 2`). In unserem Beispiel möchten wir jedoch, dass die Autoren- und Editorengruppe getrennt werden, deshalb setzen wir die Editorebene auf das nächsthöhere freie Bitmuster dezimal 2 (binär 10).

Wenn nun ein Editor eine Seite anfordert, die mit `author`-Berechtigung geschützt ist, wird der Zugriff verweigert.

- erforderliche Ebene (`author`): 1
- aktuelle Ebene (`editor`): 2
- logisches UND von 1 und 2 (`1 & 2`) ist 0, was nicht der erforderlichen Ebene entspricht

Die nächsthöhere Ebene (die dritte Bitposition von rechts) ist `supervisor`, welche die Rechte von `editor` und `author` übernimmt. Dies bedeutet, dass die Bits 1 und 2 im Supervisor-Bitmuster gesetzt werden müssen. Außerdem setzen wir das 3. Bit (2 hoch 2, bzw. 4): `1 | 2 | 4` – dies ergibt 7.

Überprüfen wir die Richtigkeit:

- erforderliche Ebene (`editor`): 2
- aktuelle Ebene (`supervisor`): 7
- logisches UND von 7 und 2 (`7 & 2`) ist 2, welches der erforderlichen Ebene entspricht.

Nun bleibt nur noch die Administratorebene übrig. Diese soll wiederum alle Rechte der Ebenen unter ihr erben: `7 | 8` (8 wird zum Setzen des 4. Bits verwendet). Das Ergebnis ist 15. Machen wir die Gegenprobe:

- erforderliche Ebene (`supervisor`): 7
- aktuelle Ebene (`admin`): 15
- logisches UND von 15 und 7 (`15 & 7`) ist 7, welches der erforderlichen Ebene entspricht.

Damit sind unsere Berechtigungen wie folgt definiert:

```
define("PHPLIB_PERM_AUTHOR",          1 | 0);
define("PHPLIB_PERM_EDITOR",          1 | 1);
define("PHPLIB_PERM_SUPERVISOR",      1 | 2 | 4);
define("PHPLIB_PERM_ADMIN",           1 | 2 | 4 | 8);

var $permissions = array(
        "author"          => PHPLIB_PERM_AUTHOR,
        "editor"          => PHPLIB_PERM_EDITOR,
        "supervisor"      => PHPLIB_PERM_SUPERVISOR,
        "admin"           => PHPLIB_PERM_ADMIN
         );
```

6.4 Zusammenfassung

In diesem Kapitel haben Sie die Basisklassen und die grundlegende Verwendung von PHPLib kennengelernt. Sie haben gesehen, dass es ein sehr leistungsfähiges Werkzeug ist, das Ihnen bei vielen Problemen helfen kann, auf die Sie bei der Erstellung von Webanwendungen unweigerlich treffen. Trotzdem ist es einfach zu verwenden, nachdem Sie den notwendigen Rahmen eingerichtet haben. Außerdem bietet es eine vollständige Infrastruktur für die Session-Verwaltung und Authentisierung von Benutzern.

Im nächsten Kapitel zeigen wir Ihnen den Einsatz von PHPLib in einer realen Anwendung. In diesem Kapitel werden wir auch eine weitere Klasse von PHPLib einführen, die wir bisher noch nicht erwähnt haben: die Klasse `Template`, mit der Sie Programmcode und Layout trennen können.

7 Anwendungen der Spitzentechnologie

Wenn Sie feststellen, dass sich alle Dinge ändern,
werden Sie an nichts mehr festhalten.
Wenn Sie keine Angst vorm Sterben haben,
gibt es nichts, was Sie nicht erreichen können.

In diesem Kapitel werden wir uns eingehender mit den Aspekten moderner Webanwendungen befassen.

Im ersten Abschnitt »Wissensrepräsentation« erzeugen wir eine Tipp-Datenbank für Benutzerbewertungen, Trefferzähler und eine unbegrenzte Anzahl von geschachtelten Kategorien. Sie erfahren etwas über Baumstrukturen und können das in Kapitel 2 »Erweiterte Syntax« erworbene Wissen in die Praxis umsetzen.

XML (Extensible Markup Language) hat sich zum am weitesten verbreiteten Standard für den Datenaustausch entwickelt. Trotzdem erschweren allgemeine Bemerkungen, wie »XML ist ein HTML mit dem Sie Ihre eigenen Tags erzeugen«, das tatsächliche Konzept dahinter zu verstehen. Wir werden versuchen, dieses zu erläutern und Ihnen eine umfassende Einführung in die XML-Analyse mit Expat, der Dokumentenobjektmodell-Schnittstelle (DOM) und LibXML zu geben.

WDDX (Web Distributed Data eXchange) bietet Mittel, um Programmiersprachenstrukturen (Objekte, Klassen, Arrays usw.) über das Internet auszutauschen. Wir erläutern, warum dies sinnvoll ist, und zeigen, wie Sie dies in Ihrer eigenen Anwendung realisieren können.

7.1 Wissensrepräsentation

In Firmenumgebungen hat sich in den letzten Jahren ein klarer Trend abgezeichnet: weg von der produktbasierten Planung, hin zu der kundenorientierten Strategie. Mit diesem Trend erlangte eine neue Technologie große Beliebtheit und Erfolg: die Wissensrepräsentation.

Für eine Firma, die einen strategischen Vorteil gegenüber ihren Mitbewerber haben möchte, ist es erforderlich, das Firmenwissen auf eine Weise zu organisieren, in der es für jedermann jederzeit leicht zugänglich ist. Mit dem Aufkommen von Firmen-Intranets wurde dieses Thema aktueller denn je.

In herkömmlichen Intranets sind Informationen häufig schwer zu finden, da sie auf vielen verschiedenen Seiten verteilt sind und von unterschiedlichen

Quellen kommen. Die tatsächlich vorhandenen Informationen sind häufig unbrauchbar, da sie nicht indiziert und nicht in kleinere logischere Einheiten unterteilt sind, was die Suche erschwert.

Wie kann eine Firma diese Probleme effizient lösen? Der Schlüssel liegt in der richtigen Wissensverwaltung. Viele Firmen bieten hierfür anspruchsvolle Lösungen. Sie sollten aber auch in Betracht ziehen, ob Sie Ihre eigenen Werkzeuge entwickeln, die einfacher und daher leichter zu verwenden sind als kommerzielle Lösungen und besser an Ihre Firmenstrategie angepasst sind.

Wir haben einen Startpunkt für Sie. Auf der CD-ROM zu diesem Buch finden Sie den vollständigen Quellcode für eine Wissensdatenbank, die sich leicht in eine unterstützende Datenbank oder ein Firmenlinkverzeichnis umwandeln lässt. Das System wurde ursprünglich für Zend Technologies entwickelt, aber die Firma erlaubte uns freundlicherweise, es mit diesem Buch zu verbreiten.

Die Anwendung verfügt über vielfältige Funktionen: Volltext-Suche, eine unbegrenzte Anzahl an Kategorien und Unterkategorien, einen Bericht, der alle Hinweise eines bestimmten Autors oder der Autoren mit den meisten Übertragungen enthält, Benutzerdaten von Einträgen, Einträge des Benutzers zur Weiterleitung von Daten und mehr.

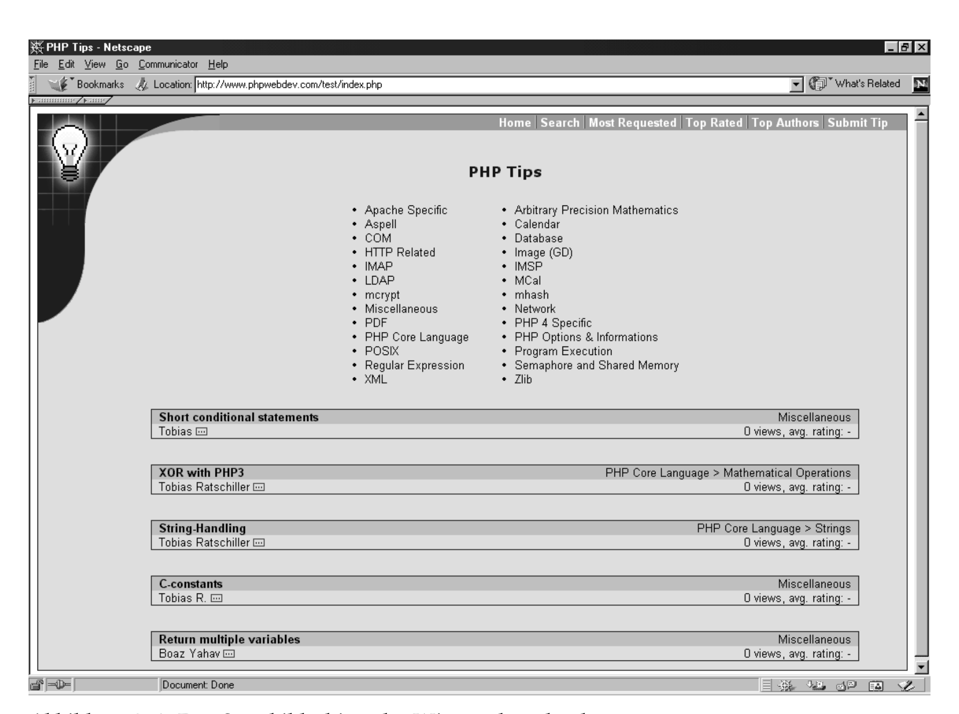

Abbildung 7.1: Der Startbildschirm der Wissensdatenbank

Das System wurde mit PHPLib (zur Datenbankabstrahierung) und HTML-Schablonen realisiert. Mit den folgenden Erörterungen erhalten Sie einen umfassenden Überblick über die Anwendungsentwicklung mit PHPLib. Abbildung 7.1 zeigt den Startbildschirm der Anwendung.

7.1.1 Anforderungskatalog

Wie in Kapitel 3 »Anwendungsdesign: ein Praxisbeispiel« besprochen, beginnt ein Projekt mit dem Aufstellen eines Anforderungskatalogs. In der Regel ist dies ein iterativer Prozess, der in enger Zusammenarbeit mit dem Kunden durchgeführt wird und von einem Systemanalytiker, Projektleiter, Berater, häufig auch Programmierern, geleitet wird. Die Analyse des Problemfeldes und das Erstellen eines Anforderungskatalogs gehört zu den wichtigsten Phasen der Softwareentwicklung, die einen großen Einfluss auf den Erfolg des Projekts haben. Dieses Projekt wurde mit einer Anforderungsliste von Zend Technologies gestartet.

Es sollte ein Softwaresystem entwickelt werden, das so aufgebaut ist, dass die PHP-Fakten, Tipps und Hinweise leicht zu finden sind. Die erste Seite der Anwendung soll die verfügbaren Kategorien unterhalb der Stammkategorie und eine Liste der in die Datenbank neu hinzugefügten Einträge zeigen. Durch Klicken auf die Kategorien soll der Benutzer die Einträge zu dieser Kategorie durchblättern können. Durch Klicken auf einen Eintrag soll er zu der Seite gelangen, die Einzelheiten zu diesem Eintrag enthält: die Überschrift zu dem Eintrag, den vollständigen Text, Namen des Autors, das Datum, an dem der Eintrag hinzugefügt wurde, und die aktuelle Beurteilung. Es sollte auch die Möglichkeit geben, auf dieser Seite Einträge mittels einer Klassifikation von Eins bis Fünf zu graduieren, wobei Eins den höchsten Grad darstellt.

Das Programm soll eine Volltext-Suchfunktion haben, bei der als Standardverknüpfungsoperator UND verwendet wird. Wenn der Benutzer »imap connect« eingibt, soll das System alle Einträge in der Datenbank zurückgeben, die im Titel oder Textkörper sowohl den Eintrage »imap« als auch den Eintrag »connect« enthalten. Bei der Suche soll Groß- und Kleinschreibung nicht unterschieden werden.

Es soll jedoch möglich sein, alle Einträge anzuzeigen, die von einem bestimmten Autor stammen. Zusätzlich sollen drei Berichte verfügbar sein, welche die Autoren mit den meisten Einträgen in der Datenbank zeigen, die Einträge mit der höchsten Bewertung und die Einträge, auf die am häufigsten zugegriffen wird.

Nur registrierte Benutzer sollen neue Einträge vorlegen dürfen. Eingereichte Einträge sollen nicht sichtbar sein, sondern in der Datenbank mit einem Flag versehen eingefügt werden, das anzeigt, dass diese Einträge genehmigt werden müssen. Sobald ein neuer Eintrag eingereicht wird, soll der Administrator benachrichtigt werden.

Auf der Zend.com-Website wird PHPLib bereits verwendet. Daher soll das System PHPLib für die Session-Verwaltung, den Datenbankzugriff und Schablonen verwenden. Zur Trennung von Code und Layout soll die PHPLib-Klasse `Template` verwendet werden. Außerdem sollte das System eine saubere API haben, da es später bei Zend Technologies von verschiedenen Entwicklern gepflegt wird.

Für gewöhnlich liefert der Kunde keinen so detaillierten Anforderungskatalog. Häufig wissen Kunden nicht, wie sich ihre geschäftlichen Anforderungen in der Software realisieren lässt. Der Kunde ist kein Experte der Softwareentwicklung, sondern kennt nur das Problemfeld. Während der ersten Besprechungen mit dem Kunden sollte der Systemanalytiker aus dem dargelegtem Problemfeld einen Anforderungskatalog erstellen. »Wozu dient die Anwendung?« und »Was soll die Anwendung für den Benutzer tun?« sind typische Fragen für diese Phase. Es ist die Aufgabe des Analytikers, dem Kunden zu helfen, die Probleme so zu formulieren, dass sie sich in Softwarelösungen umsetzen lassen. Während der Analysephase lernt der Analytiker das Problem genauer kennen und kann es in konkrete und dokumentierbare Begriffe fassen.

7.1.2 Spezifikation

Der Anforderungskatalog vermittelt Ihnen ein allgemeines Verständnis über das Problem. Sobald Sie dieses haben, sollten Sie Richtlinien für das tatsächlich zu entwickelnde Programm aufstellen, d.h. eine Spezifikation schreiben. Dazu müssen Sie zunächst herausfinden, welche Datenstukturen Sie benötigen.

Versuchen Sie, das komplexe Problem in kleinere Einheiten zu teilen. Wenn Sie den Anforderungskatalog und das Problemfeld analysieren, stellen Sie fest, dass es drei wichtige Datenstrukturen gibt und der Rest der Anwendung auf diesen aufbaut. Die wichtigste Struktur bildet ein Eintrag in der Wissensdatenbank. Was ist ein Eintrag? Aus dem Anforderungskatalog wissen wir, dass einem Eintrag Eigenschaften zugeordnet sind: Titel, Hauptteil, Autor, Kategorie, Beurteilung und Protokolle. Da wir bereits ähnliche Probleme gelöst haben, sehen wir ein klares Muster in dieser Datenstruktur: Es handelt sich um einen einfachen Behälter. Wir wissen aber auch, dass wir eine Methode brauchen, um auf diesen Behälter zu referenzieren. Unserer Erfahrung nach ist dies sehr wichtig und kann einen wirklich guten Programmierer von einem mittelmäßigen unterscheiden. Das schwierigste Problem lässt sich leicht lösen, wenn Sie es früher schon einmal gelöst haben.

Eine Datenstruktur für die Kategorie zu generieren, geschieht nach denselben Prinzipien, wobei wir anfänglich nur wissen, dass dieser Struktur die Eigenschaft »Name« zugeordnet werden soll. Laut Anforderungskatalog brauchen

wir verschachtelte Kategorien. Deshalb muss diese Struktur auch einen eindeutigen Bezeichner haben (zwei Kategorien in verschiedenen Zweigen könnten denselben Namen haben). Eine weitere Anforderung lautet: unbegrenzte Verschachtelung. Aber dies überspringen wir für den Moment, da es ein separates Problem ist.

Die dritte Datenstruktur ist bereits definiert, da wir PHPLib verwenden. Sie entspricht der PHPLib-Klasse `Auth`.

Dieser Ansatz unterscheidet sich von der herkömmlichen Top-Down-Programmierung und funktionalen Zerlegung. Bei der funktionalen Zerlegung werden die Funktionen eines zu erstellenden Systems gekennzeichnet (in unserem Beispiel »Strukturierung von Fakten in Kategorien«, »Bereitstellung von Berichten für die Einträge mit den meisten Zugriffen« usw.) und in kleinere Unterfunktionen unterteilt, bis die Funktionen atomar vorliegen und sich direkt in Programmfunktionen abbilden lassen. Zu diesem Zeitpunkt unternehmen wir in dieser Richtung noch nichts, da wir noch gar nicht wissen, wie Unterfunktionen definiert werden. Funktionale Zerlegung hört sich großartig an – bis Sie es ausprobiert haben. Sie kann Sie in eine vollkommen falsche Richtung führen, und sobald Sie einmal angefangen haben, ist es nahezu unmöglich, Entscheidungen zu korrigieren, da Sie die Funktionen nur immer wieder teilen können – Sie müssen also komplett von vorn beginnen.

Statt dessen versuchen wir, das gesamte Problem in Entwurfsmuster zu unterteilen. Wir versuchen, Probleme zu erkennen, für die wir bereits eine Lösung entwickelt haben. Die Lösung für ein ähnliches Problem zu kennen, ist die beste Ausgangssituation, um ein neues Problem zu lösen. Für das Problem der Authentisierung brauchen wir beispielsweise keine Art von funktionaler Zerlegung vornehmen, da wir wissen, dass wir einfach Teile von PHPLib dafür verwenden können.

Die Struktur für die Einträge und Kategorien können bereits auf den Programmcode abgebildet werden. Unsere Anwendung hält die Eintrags- und Kategoriestrukturen in Klassen fest:

```
class category
{
    var $cat_id, $cat_name, $parent_id;
}

class entry extends category
{
    var $entry_id, $title, $body, $t_stamp, $author, $views, $votes,
$rating;
}
```

Klassen zu verwenden, war eine Entscheidung, die durch den Entwurf entstand. Sie war nicht durch den Anforderungskatalog bestimmt. Unsere bisherige Erfahrung hat uns einfach gezeigt, dass die Verwendung von Klassen zu einem saubereren Programmcode führt, da Sie mehrere verschiedene Instanzen verwenden können.

Sie können diese Datenstrukturen als unterschiedliche Bereiche der Software ansehen. Sie bilden logische Einheiten, die aber miteinander interagieren. Ziel der Spezifikation ist, alle Bereiche in einer Anwendung abzudecken. Wie die Bereiche im Programmcode dargestellt werden, ist an dieser Stelle nicht von Bedeutung und dient nur zur Verdeutlichung der Strukturen.

Die Übertragung dieser Datenstrukturen in ein relationales Datenbankmodell ist recht einfach. Die Anwendung verwendet Datenbanktabellen, um Einträge in der Wissensdatenbank zu speichern: verfügbare Kategorien, Eintragsbewertungen und Zugriffsprotokolle. Die beiden Haupttabellen sind `entries` und `categories`, die Links zu den Untertabellen `ratings` und `logs` haben. Da MySQL keine fremden Schlüssel kennt, müssen diese Verknüpfungen von der Anwendung selbst verarbeitet werden. Wenn der Administrator beispielsweise einen Eintrag löschen möchte, muss er auch die entsprechenden Einträge (referenziert mit derselben `entry_id`) in den Tabellen `ratings` und `logs` löschen. Abbildung 7.2 zeigt ein Diagramm, das die Beziehung der Einheiten in der Tabellenstruktur zeigt.

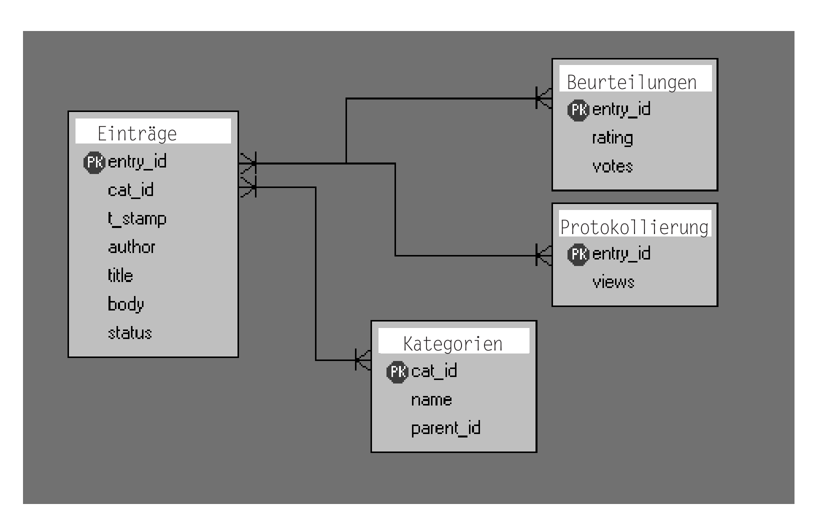

Abbildung 7.2: Einheiten-Beziehungsdiagramm (ERD) für die Wissensdatenbank

Nachdem Sie die grundlegenden Datenstrukturen der Anwendung kennen, stellt sich als Nächstes die Frage, was mit ihnen passiert. Der Anforderungskatalog nennt eine Reihe von Aktionen, welche die Anwendung zulassen soll. Es ist nun unsere Aufgabe, diese Aufgaben sauber zu trennen.

In der Regel werden die Aktionen unter den einzelnen Datenstrukturen zusammengefasst, die zuvor definiert wurden. Konzentrieren wir uns zunächst auf die Aktionen, die sich mit Einträgen in der Datenbank befassen:

- Auslesen eines speziellen Eintrags. Für diese Aktion muss der Bezeichner des auszulesenden Eintrags bekannt sein. Die Aktion schlägt fehl, wenn es in der Datenbank keinen Eintrag gibt, der mit dem Bezeichner übereinstimmt. Sie kann auch fehlschlagen, wenn Systemfehler auftreten, z.B. wenn es keinen Zugriff auf das Datenbanksystem gibt. Wenn die Aktion erfolgreich ist, wird die Struktur für den Eintrag zurückgegeben.
- Auslesen von Einträgen in einer spezifischen Kategorie. Für diese Aktionen muss der Name der Kategorie bekannt sein, für welche Einträge ausgelesen werden sollen. Wenn die Aktion erfolgreich ist, wird eine Liste der gültigen Einträge zurückgegeben. Eine Liste der Einträge? Moment – solche Datenstrukturen hatten wir bisher noch nicht definiert. Es wird Zeit, zur Spezifikation zurückzukehren und eine neue Struktur für Listen anzulegen.
- Auslesen der zehn zuletzt hinzugefügten Einträge
- Auslesen der Einträge mit der höchsten Bewertung
- Auslesen der Einträge, auf die am meisten zugegriffen wird
- Auslesen alle Einträge von einem bestimmten Autor

Diese Liste lässt sich für alle genannten Aktionen und Bereiche fortsetzen. Am Schluss erhalten wir eine umfassende Liste aller erforderlichen Strukturen und Aktionen, die das gesamte Projekt dokumentiert.

Um es zusammenzufassen: Wir haben einen Anforderungskatalog erstellt, indem wir das Problemfeld umrissen und die Funktionen beschrieben haben, welche die Anwendung haben sollte. Indem wir den Anforderungskatalog in konkretere Begriffe gefasst haben, haben wir eine Spezifikation erstellt. Die Spezifikation beschreibt die Datenstrukturen und das Verhalten der Anwendung.

Nun sollten wir uns im Einzelnen damit befassen, wie die Anwendung realisiert werden kann. Wir picken uns hier zwei interessante Aspekte heraus, nämlich die Verwendung von Schablonen und den Einsatz von verschachtelten Kategorien in SQL.

7.1.3 Die Klasse Template

Wie wir in Kapitel 6 »Datenbankzugriff mit PHP« beschrieben haben, bietet PHPLib eine Lösung für viele Probleme, die bei Webanwendungen auftreten. In unserem Fall benutzen Zend-Entwickler PHPLib bereits für Teile ihrer Website, so dass wir es als Standard für die Session-Verwaltung, Datenbankabstrahierung und HTML-Schablonen verwendet haben.

Die PHPLib-Klasse `Template` ermöglicht eine Trennung von Programmcode und Layout, ähnlich wie die Klasse `EasyTemplate`, die wir in Kapitel 5 »Grundlegende Webanwendungsstrategien« entwickelt haben. Diese Klasse hat eine größere Vielfalt an Funktionen, als die von uns entwickelte Klasse. Sie kann beispielsweise Blöcke enthalten, die Abschnitte markieren, die mehr als einmal ersetzt werden sollen (was z.B. in Tabellenzeilen nützlich ist). Außerdem kann sie in einer Instanz mehrere Dateien öffnen und diese sehr einfach kombinieren. Der Nachteil ist, dass sie sich nicht so intuitiv verwenden lässt wie `EasyTemplate`.

Die Klasse `Template` ist vollständig unabhängig von dem Rest von PHPLib, so dass Sie diese auch ohne andere PHPLib-Funktionen einsetzen können. Wenn Sie sich den Quellcode für diese Klasse ansehen möchten, finden Sie ihn in der Datei `template.inc` der PHPLib-Distribution.

Ähnlich wie `EasyTemplate` speichert PHPLibs Schablonenklasse HTML in separate Dateien, und verwendet Platzhalter für Daten, die dynamisch von PHP ersetzt werden sollen. Die skalaren Platzhalter, die durch einfache Zeichenketten ersetzt werden, haben dasselbe Format wie jene in `EasyTemplate`.

Listing 7.1 zeigt den Programmcode, der diese einfache Schablone verarbeitet.

```
// Create a template instance
$tpl = new Template ();

// Load file, assign an identifier to it
$tpl->set_file("page" => "basic_template.inc.html");

// Assign contents to the placeholders
$tpl->set_var(array("TITLE" => "This is a Template test",
                    "CONTENTS" => "Hello World!"));

// Parse into a temporary variable (identifier)
$tpl->parse("out", "page");

// Output the parsed template
$tpl->p("out");
<html>
 <title>{TITLE}</title>
 <body>
  {CONTENTS}
 </body>
</html>
```

Listing 7.1: Einfaches Beispiel der Klasse Template

Die erste Zeile erzeugt eine Instanz der Klasse `Template`. Der Konstruktor der Klasse hat zwei optionale Argumente. Das erste optionale Argument gibt ein Basisverzeichnis an, in dem sich Ihre Schablonendateien befinden. Standardmäßig ist dies das aktuelle Verzeichnis `./`. Das zweite Argument definiert, wie Platzhalter behandelt werden, die nicht in Ihrem Skript verwendet werden. Die möglichen Werte sind `keep`, `comment` oder `remove`, wobei der Standardwert `remove` ist. Bei `keep` bleiben Platzhalter erhalten. Wenn in unserem Beispiel dem Platzhalter `{TITLE}` kein Wert zugewiesen wäre, würde der Platzhalter später in der analysierten Schablone unverändert erscheinen. Wenn die Variable auf `comment` gesetzt wäre, sähe die Ausgabe unseres Beispiels folgendermaßen aus, falls dem Platzhalter `{TITLE}` kein Wert zugewiesen ist:

```
<html>
 <title><!-- Template : Variable TITLE undefined --></title>
 <body>
  Hello World!
 </body>
</html>
```

Bei `remove` (der Standardeinstellung) werden nicht zugewiesene Platzhalter ohne Nachfrage aus der Schablone gelöscht.

In der zweiten Zeile unseres Beispiels wird eine Schablonendatei mithilfe von `set_file()` einer Klasse zugeordnet. Diese Funktion hat zwei Argumente. Im ersten Argument wird ein Handle angegeben, mit dessen Hilfe die Schablonendatei in künftigen Funktionen referenziert wird. Das zweite Argument enthält den Dateinamen. Bedenken Sie, dass die Datei in jenem Pfad gesucht wird, der im Konstruktor angegeben wurde (in unserem Beispiel das aktuelle Verzeichnis). Sie können der Funktion `set_file()` auch ein assoziatives Array übergeben, um mehrere Dateien auf einmal zuzuweisen. In diesem Fall sind die Schlüssel des Array die Handle, und die Elemente definieren die einzelnen Dateinamen.

Anschließend werden den Platzhaltern der Schablone mithilfe von `set_var()` Zeichenketten zugeordnet. Auch hier können Sie wieder ein einzelnes Schlüssel / Wert-Paar an die Funktion übergeben oder ein assoziatives Array zur Stapelverarbeitung. Die übrigen Zeilen unseres Beispiels rufen die Analysefunktion (`parse()`) auf und geben das Ergebnis aus (`p()`).

Dieses Beispiel zeigt eine sehr einfache Verwendung der Klasse `Template`. Es arbeitet mit einer einzigen Schablonendatei und ersetzt jeden Platzhalter durch eine Zeichenkettenvariable. Hier hätten Sie auch `EasyTemplate` verwenden können, was wahrscheinlich schneller und intuitiver gewesen wäre als der PHPLib-Ansatz. Wie leistungsfähig die Klasse `Template` wirklich ist, zeigt sich, wenn Sie diese in komplexeren Szenarien einsetzen.

Eine ihrer Stärken besteht darin, dass die Klasse `Template` mehrere Vorlagendateien verarbeiten und diese in eine Ausgabedatei zusammenfassen kann. Die Wissensdatenbank macht extensiven Gebrauch von dieser Funktion. Eine Seitenschablone definiert das allgemeine Layout und die Struktur, die Cascading Style Sheets sowie Kopf- und Fußzeile, während viele kleinere Schablonen den Inhalt innerhalb der »Eltern«schablone festlegen. Betrachten Sie die folgenden Auszüge aus der Hauptseite der Anwendung `index.php`:

```
$tpl->set_file(array(
     "page" => "page.inc.html",
     "table" => "table.inc.html",
     "entry_summary" => "entry_summary.inc.html"
     ));

// [rest of code, assignments, etc]

$tpl->parse("CONTENTS", "table", true);
$tpl->parse("CONTENTS", "entries", true);
$tpl->parse("CONTENTS", "page");
$tpl->p("CONTENTS");
```

Die Hauptschablone wird mit dem Bezeichner `page` referenziert. Hierbei handelt es sich im Wesentlichen um einen HTML-Frame, der einen wichtigen Platzhalter, nämlich `{CONTENTS}`, für den eigentlichen Inhalt der Seite enthält. Dieser Platzhalter wird durch eine weitere separate Schablonendatei ersetzt, die als `table` referenziert wird. Diese Vorgehensweise funktioniert, weil die Klasse `Template` Ihnen ermöglicht, die Ergebnisse einer Analyse an einen Platzhalter anzuhängen. Das Skript analysiert zunächst die Schablonendatei, die mit `table` referenziert wird, und hängt sie anschließend an die Hauptschablonen an.

Wenn wir uns eine andere Schablonendatei der Anwendung, nämlich `entry_summary.inc.html`, ansehen, können wir eine weitere höhere Funktion der Klasse `Template` sehen: *Blöcke*. Dynamische Blöcke werden für Teile einer Schablone verwendet, die iterativ durch sich selbst ersetzt werden. In unserem Fall wird diese Funktion verwendet, um die letzten fünf Einträge der Wissensdatenbank anzuzeigen. Die Schablone `entry_summary.inc.html` enthält einen Block, der wiederholt wird, um fünf zusammen gefasste Einträge zu erzeugen. Ein Block wird in der Schablone mithilfe einer Kommentarsyntax definiert.

```
<!-- BEGIN blockname -->
   block
<!-- END blockname -->
```

Im Programmcode wird auf den Block mithilfe der Funktion `set_block()` zugegriffen. Das erste Argument dieser Funktion gibt die Elternreferenz an (in der Regel ein Verweis auf die Schablonendatei). Das zweite Argument ent-

hält den Namen des Blocks. Im optionalen dritten Argument kann der Name einer neuen Referenz angegeben werden. Wenn dieser fehlt, geht die Funktion davon aus, dass Sie mit dem Blocknamen identisch ist. In unserem Beispiel würde `set_block()` folgendermaßen aufgerufen werden:

```
set_block("table", "blockname");
```

Die erzeugte Referenz (`blockname`) kann dann wie Referenzen gehandhabt werden, die mit `set_file()` erzeugt wurden, und normal analysiert werden. Zugegeben, es klingt verwirrend, wenn Sie dies zum ersten Mal hören. Sehen wir uns an, wie die Klasse `Template` vom Konzept her funktioniert. Eine wichtige logische Einheit der Klasse `Template` bilden die `Handles`. Handles ähneln Linkbezeichnern (Ressourcen-IDs): sie zeigen auf einen bestimmten Datensatz und können in verschiedenen Funktionen als Referenzen für diesen Datensatz verwendet werden. Sie können Handles auf eine der folgenden drei Arten erzeugen:

- `set_file()` erzeugt einen Handle für eine Schablonendatei
- `set_var()` erzeugt einen Handle für einen Platzhalter innerhalb einer Schablone
- `set_block()` erzeugt einen Handle für einen Block innerhalb einer Schablone

Mit jeder dieser Funktionen können Sie den zu erzeugenden Handle angeben. In `set_file()` und `set_var()` wird der zu erzeugende Handle im ersten Argument angegeben (bzw. im Schlüssel des Array, wenn als Argument ein assoziatives Array übergeben wird). In `set_block()` wird der Handle im zweiten Argument angegeben. In Funktionen, wie`parse()`, `subst()` oder `get_undefined()`, verwenden Sie den zuvor erzeugten Handle, um auf den Datensatz zu referenzieren, den die Funktionen verarbeiten sollen. Für die Funktionen ist es unerheblich, wie der Handle erzeugt wurde. Sie arbeiten sowohl mit ganzen Schablonendateien als auch mit Platzhaltern oder dynamischen Blöcken. Sehen wir uns ein weiteres einfaches Beispiel an. Angenommen, Sie haben eine Schablonendatei mit einem Platzhalter und einem dynamischen Block:

```
<b>{PLACEHOLDER}</b>
<ul>
<!-- BEGIN block -->
   <li> {BLOCK_PLACEHOLDER}
<!-- END block -->
</ul>
```

Um die Datei zu analysieren, müssen Sie zunächst mit `set_file()` einen Handle für die gesamte Datei definieren. Der normale Platzhalter lässt sich ganz normal verarbeiten, wie wir früher schon gezeigt haben. Anschließend definieren wir einen Block-Handle. Diesen Block können Sie nun genauso behandeln wie den Datei-Handle selbst. Er ist ein genauso wichtiger unab-

hängiger Teil innerhalb der Datei. Dadurch können Sie auch zwei Handles kombinieren, wie wir es bereits mit zwei separaten Dateien getan haben. Im Programmcode sieht dies folgendermaßen aus:

```
$tpl->set_file("page", "page.inc.html");

// Assign value to scalar placeholder
$tpl->set_var("PLACEHOLDER", "This is just a test.");

// Create block handle, named "block"
$tpl->set_block("page", "block");

// Create three block instances
for($i=0; $i<3; $i++)
{
    // Replace placeholder for this loop iteration
    $tpl->set_var("BLOCK_PLACEHOLDER", "Loop #$i");

    // Parse block, append the result to itself
    $tpl->parse("block_handle", "block", true);
}

// Parse and output page
$tpl->parse("page", "page");
$tpl->p("page");
```

Auf die Weise erhält der Programmdesigner die Möglichkeit, Zeilenschablonen zu definieren, ohne sich um den PHP-Code kümmern zu müssen. Es macht zwar die Aufgabe des Entwicklers flexibler, trotzdem gibt es noch einige Szenarien, in denen Sie keine andere Möglichkeit haben, als Code und Layout zu mischen. Ein Beispiel hierfür ist die Suchergebnisseite unserer Anwendung. Im Programmcode für diese Seite finden Sie folgenden Abschnitt:

```
$entries = kb_get_entries_by_keyword($keywords);
// Any entries found?
if($entries)
{
    $tpl->set_block("tip_summary", "tip", "entries");
    kb_entries_to_template($entries, $tpl);

    $tpl->set_var(array(
            "RESULTS_TITLE" => sprintf(count($entries).
            ➥" %s found:", count($entries) > 1 ? "entries" : "entry"),
            "KEYWORDS" => $keywords
            ));
}
```

```
else
{
    $tpl->set_var("MESSAGE",  '<div align="center"><i>No entries _
               ➥found.</i></div>');
    $tpl->parse("entries", "tip", true);
}
```

Der Programmcode überprüft, ob Einträge in der Datenbank gefunden wurden, die mit dem Suchbegriff übereinstimmen. Entsprechend wird entweder eine Meldung ausgegeben, die besagt, dass keine Einträge gefunden wurde, oder die Liste der gefundenen Einträge. Im Listing passt der Code auch die Meldung an, je nach dem ob ein Eintrag oder mehrere Einträge gefunden wurden (»1 entry found« bzw. »`x` entries found«). Dies ist ganz klar ein Layout-Problem, denn die Anzahl der gefundenen Einträge beeinflusst in keiner Weise die Programmlogik. In einer idealen Umgebung wäre der Entwickler selbst in der Lage, diese Meldungen bereitzustellen. Vielleicht möchte der Entwickler die Meldung »No entries found« rot und fett gedruckt und die Anzahl der gefundenen Einträge groß und fett gedruckt ausgeben. In unserem Fall müsste der Entwickler den Programmierer bitten, den Code entsprechend zu ändern. Spätestens nach der dritten Änderung wird dies sowohl für den Entwickler als auch für den Programmierer sehr nervenaufreibend.

Ein Ansatz für die Lösung dieses Problems besteht darin, der Schablone einen Teil der Kontrolle zurückzugeben, und den Entwickler über die Schablonenlogik entscheiden zu lassen. Die Schablonen enthielten dann eine einfache Metaskriptsprache, die folgendermaßen aussieht:

```
{{if ENTRIES_FOUND > 1}}
    {{ENTRIES_FOUND}} entries found:
{{/if}}
{{if ENTRIES_FOUND=1}}
    One entry found:
{{/if}}
{{if ENTRIES_FOUND=0}}
    No entries found for your search!
{{/if}}
```

Natürlich existiert ein fließender Übergang zwischen Trennung von Code und Layout und Vermischung dieser beiden. Möchten Sie Layout im Programmcode oder Programmcode im Layout haben? Dies ist die klassische Frage vom Ei und der Henne. Zum Zeitpunkt, als dieses Buch erstellt wurde, gab es erste Bemühungen, eine Schablonen-API für die PHP-Standarddistribution zu erstellen. Das obige Metaskriptbeispiel stammt aus dem Entwurf von Andrei Zmievski für eine Schablonensprache. Andrei (der ein Befürworter der Schablonen-API und Entwickler des PHP-Kerns ist) plant, eine Reihe von weiteren Funktionen zu implementieren, z.B. die vordefinierten Standardvariablen `#ODD` und `#EVEN` in dynamischen Blöcken. Damit ließen sich die beliebten Farb-

änderungen in wiederholten Tabellenzeilen realisieren, die ansonsten vom Programmierer eingefügt werden müssten. Darüber hinaus möchte Andrei die Schablonen-API direkt in PHP integrieren, was einige Vorteile gegenüber heutigen Schablonenlösungen, wie jenen von PHPLib, bringen würde. Zunächst einmal wäre es Standard, und Softwareentwickler könnten sich darauf verlassen. Da es direkt im PHP-Kernprogramm integriert würde, könnte dies auch zu einer erheblichen Leistungssteigerung führen. Die analysierte Schablonen könnten beispielsweise im Speicher gecacht werden.

7.1.4 Rekursion mit SQL

Unsere Anwendung ermöglicht eine unbegrenzte Verschachtelung von Kategorien. Wir haben uns für die einfachste und am leichtesten zu realisierende Lösung zur Verschachtelung von Kategorien entschieden. Die Tabelle `categories` wird wie folgt definiert:

```
CREATE TABLE categories (
   cat_id bigint(21) DEFAULT '0' NOT NULL auto_increment,
   name varchar(32) NOT NULL,
   parent_id bigint(21) DEFAULT '0' NOT NULL,
   PRIMARY KEY (cat_id),
   KEY parent_id (parent_id)
);
```

Verantwortlich für die Verschachtelung ist natürlich das Feld `parent_id`. Es enthält den Wert `cat_id` der Kategorie eine Ebene darüber. Genau genommen ist dies die einfachste Form für eine Verzeichnisstruktur. Jeder Knoten hat genau eine Eigenschaft, die auf den Elternknoten referenziert. Dieser Ansatz hat jedoch eine Reihe von Nachteilen. Der größte ist wohl, dass es unmöglich ist, für einen Knoten alle Elternknoten in einer SQL-Abfrage zu ermitteln. Dazu brauchen Sie mehrere SQL-Abfragen. Genau genommen brauchen Sie `n-1` Einzelabfragen bei einer Verschachtelung von `n` Ebenen..

Wir haben uns für eine rekursive Funktion entschieden, um die Elternknoten mit der Funktion `kb_cat_get_parents()` zu ermitteln. Sie liest die Kategorienknoten aus der Datenbank aus, solange `cat_id` mit der Stammkategorie übereinstimmt. Dies lässt sich am besten anhand eines Beispiels demonstrieren. Angenommen, es gibt drei verschachtelte Kategorien:

```
INSERT INTO categories VALUES (1, 'Main Category', 0);
INSERT INTO categories VALUES (2, 'Sub Category I', 1);
INSERT INTO categories VALUES (3, 'Sub Category II', 2);
```

Beim Aufruf von `cat_id` mit dem Anfangswert `3`, liest die Funktion `kb_cat_get_parents()` zunächst die Eltern-Kennnummer für diesen Knoten aus (die in unserem Beispiel `2` lautet). Anschließend ruft sie sich in einer rekursiven Funktion mit dieser Kennnummer selbst auf. Den Abschluss dieser rekursi-

ven Funktion bildet `parent_id == 0`. Dies ist die Stammkategorie, über welcher keine Knoten liegen können. Die Funktion ruft sich solange selbst auf, bis diese Bedingung erfüllt ist.

7.1.5 Authentisierung

Eine Anforderung der Anwendung ist, dass nur registrierte Benutzer neue Einträge machen dürfen. Dank PHPLib lässt sich die Authentisierung überprüfen, indem Sie in dem Skript, das Sie schützen wollen, den Aufruf `page_open()` einfügen, in unserem Fall `submit.php`, überprüfen. Auf diese Weise kann der Benutzer auf den Seiteninhalt nur zugreifen, nachdem er authentisiert wurde.

In der Tabelle `entries` speichern wir die eindeutige Benutzerkennnummer, die von PHPLib geliefert wird. Wie wir in diesem Kapitel bereits gezeigt haben, können Sie auf diese Kennnummer über das Array `$auth` zugreifen. Sie wird in `$auth->auth["uid"]` gespeichert, während `$auth->auth["uname"]` gibt den Benutzernamen angibt.

Unsere Anwendung kümmert sich nicht um die Benutzerverwaltung. Deshalb müssen Sie irgendwo anders auf Ihrer Website Werkzeuge zur Registrierung eines Benutzers, Bearbeitung von Registrierungen und Verschicken von vergessenen Passwörtern einrichten. Da diese Aufgabe Ihnen überlassen bleibt, haben wir auch keine Möglichkeit, den vollständigen Namen eines Benutzers zu ermitteln. Alles, was wir haben, ist die eindeutige Benutzerkennnummer und den Benutzernamen. Es gibt daher eine Funktion namens `real_user_name()`, welche als Parameter die Benutzerkennnummer akzeptiert und den vollständigen Namen dieses Benutzers zurückgibt. Standardmäßig gibt diese Funktion nur die Benutzerkennnummer zurück. Sie sollten diese Funktion erweitern, um in Ihrer Datenbank den vollständigen Namen des Benutzers zu suchen und zurückzugeben.

7.1.6 Das fertige Produkt

Die Funktion `real_user_name()` wird zusammen mit allen anderen API-Funktionen in einer zentralen Datei namens `lib.inc.php3` gespeichert. Da es für alle Funktionen im Programmcode eine einfache Dokumentation zur Syntax gibt, lässt sich sehr leicht eine API-Übersicht automatisch erstellen. Hierzu benötigen wir nur den Befehl `grep`:

```
grep '^[\\\/ ]*\*' lib.inc.php
```

Die Übersicht ist jedoch kein Ersatz für eine vollständige technische Dokumentation und sollte nur als Kurzreferenz dienen. Nachdem die API definiert ist, beschränkt sich die restliche Anwendung hauptsächlich darauf, die API-Funktionen aufzurufen und die Ergebnisse auszudrucken.

7.2 PHP und XML

Hinweis: Wenn Sie bereits mit dem grundlegenden Konzept von XML vertraut sind, können Sie die nächsten Abschnitte, in denen wir eine kurze Einführung in XML geben, überspringen und direkt im Abschnitt über PHP und Expat fortfahren.

7.2.1 Was ist XML?

XML (Extensible Markup Language) ist eine Meta-Auszeichnungssprache für Dokumente, die strukturierte Informationen enthalten. Gehen wir es Satz für Satz noch einmal durch:

- XML ist *erweiterbar* (englisch: eXtensible). Nehmen wir HTML: Das Tag <h1> bezeichnet stets eine Überschrift erster Ebene. In XML hat das Tag dagegen keine Bedeutung, bis Sie ihm durch eine zusätzliche Regel (der Document Type Definition (DTD), eine Bedeutung geben.
- XML dient der *Auszeichnung* (englisch: Markup). Genau wie HTML (das zumindest theoretisch keine Layoutinformationen enthält) hat auch XML keine.
- XML ist eine Metasprache (englisch: meta Language). XML hat keine feste Anzahl von Tags, sondern besitzt eine Funktion zur Definition von Tags.
- XML arbeitet mit Dokumenten. Wohlgemerkt: Dokumente – es ist nicht begrenzt auf Dateien! Dokumente können aus einer Datenbank, über das Netzwerk oder aus Dateien kommen.
- XML definiert strukturierte Informationen. Es fasst einzelne Teile von Daten in größeren Konstrukten zusammen, gibt diesen eine Bedeutung in einem Kontext und setzt sie in eine strukturelle Beziehung.

Strukturierte Information

Es gibt ein Schlüsselkonzept, das Sie verstehen müssen, wenn Sie über XML reden: strukturierte Dokumente oder – eleganter ausgedrückt – strukturierte Informationsauszeichnung. Die strukturierte Auszeichnung definiert explizit den Aufbau und den semantischen Inhalt (die kontextuelle Bedeutung) eines Dokuments. Sie hat keinen Einfluss auf die Art und Weise, in der das Dokument für den Leser erscheint. Die Interpretation der Daten (syntaktische Analyse, Layout usw.) bleibt der verarbeitenden Anwendung überlassen. Nehmen wir z.B. das HTML-Tag <p> (paragraph = Absatz): Es kennzeichnet mehrere Sätze, die zusammengehören und eine logische Einheit bilden. Das Tag an sich sagt nichts darüber aus, wie der Ansatz im Browser wiedergegeben wird. Der Browser könnte davor oder danach eine Leerzeile einfügen, die erste Zeile einziehen oder einen Zierrahmen um den Absatz ziehen. Hier geht

es um logische Auszeichnung. Die Stilinformation wird direkt im Browser eingegeben. XML-Dokumente setzen sich aus solchen logischen Auszeichnungen zusammen. Wie bei HTML werden Tags verwendet, um die Auszeichnungsinformationen zu kennzeichnen. Bei XML gibt es jedoch keine visuellen Elemente wie bei HTML (denken Sie an <font>). Es ist auf logische Auszeichnung beschränkt. Es gibt keine Möglichkeit, in XML z.B. ein Wort kursiv zu setzen. Sie können es lediglich, gemäß seiner semantischen Bedeutung kennzeichnen, z.B. mit <emphasis>.

So weit zur Auszeichnung. Wo ist die Struktur? XML-Tags können verschachtelt werden und haben einen Kontextstatus, d.h. es ist relevant, wo sie in einem Dokument auftauchen. Die Tagkombination <chapter><title> wird anders behandelt als <book><title>. In der XML-Spezifikation gibt es keine Beschränkung in Bezug auf die Anzahl der verschachtelten Elemente. Die einzige Anforderung ist, dass alle Elemente ihrem Ursprung im Stammelement haben.

XML-Verwandte

Der Vorläufer von XML ist SGML. Seit es 1986 zum ISO-Standard wurde, wird SGML (Standard Generalized Markup Language) von größeren Firmen aller Industriezweige zur Erzeugung von strukturierten Dokumenten verwendet. SGML ist jedoch ein komplizierte Standard, der von Anwendungen kaum unterstützt wird. Die meisten SGML-Anwendungen – Editoren, Speicherserver, Konvertierungswerkzeuge – sind daher sehr teuer; häufig kosten Sie über $10.000.

HTML hat dagegen industrieweite Unterstützung, und wird auf Millionen von Websites verwendet. Es definiert einen einfachen Typ von Dokument für eine gewöhnliche Klasse von kurzen Artikeln mit Kopfzeilen, Absätzen, Listen und/oder Bildern und enthält einige Funktionen zum Einbinden von Hypertext-Marken und Multimedia-Anwendungen. Es ist jedoch in Bezug auf die Flexibilität und Erweiterbarkeit sehr begrenzt. Die Tags und die Semantik sind festgelegt. Sie können kein eigenes Tag für einen Eintrag in ein Inhaltsverzeichnis definieren. Außerdem eignet es sich nicht für andere Medien als Computerschnittstellen. Wenn Sie jemals versucht haben, Artikel auszudrucken, die an mehrere Dateien verteilt wurden, wissen Sie, was dies heißt. Die sehr offene Spezifikation von HTML führte dazu, dass sich viele verschiedene HTML-»Dialekte« entwickelt haben. Wie Sie wissen, ist es eine Kunst für sich, browserneutrales HTML zu schreiben.

So entstand die Notwendigkeit, ein neues Format zu generieren, mit dem strukturierte Dokumente im gesamten Web verwendet werden können. Um die Beschränkungen der einzigen beiden ernsthaften Alternativen HTML und SGML zu überwinden, wurde XML entwickelt.

Der Entwurf von XML sah einige klar definierte Ziele vor:

- Es sollte einfach zu verwenden sein – sowohl für Benutzer als auch für Entwickler, welche die XML-Parser erstellten. Die Komplexität von SGML bedeutete eine Beschränkung, die beseitigt werden sollte.
- XML sollte offen sein, um eine Vielzahl von Anwendungen und Unterprotokolle unterstützen zu können. Es sollte keine Abhängigkeit zu einem einzigen unflexiblen Dokumententyp entstehen wie bei HTML.
- XML sollte eine strikte Syntax haben. Optionale Funktionen können zu Kompatibilitätsproblemen führen, wenn mehrere Benutzer ein Dokument gemeinsam verwenden. Es gab die Befürchtung, dass dasselbe passiert wie bei HTML: mehrere konkurrierende und nicht kompatible Programme entstanden.
- XML sollte zu SGML kompatibel sein. Die Mitglieder des Entwicklungskomitees waren auch in den SGML-Bemühungen einbezogen und integrierten viele Daten aus SGML-Systemen.

Die Entwicklung führte zu einer klaren Spezifikation, die am 10.Februar 1998 vom World Wide Web Consortium (W3C) als Empfehlung der Extensible Markup Language (XML) 1.0 verabschiedet wurde.

XML unterscheidet sich von SGML: XML verzichtet auf viele der komplexeren und weniger verwendeten Funktionen von SGML, und erzeugt eine neue, reduzierte SGML-basierte Anwendung. Da es eine Untermenge von SGML darstellt, können Sie ein XML-Dokument mit jedem SGML-kompatiblen System lesen. Jedes gültige XML-Dokument ist ein gültige SGML-Element.

XML unterscheidet sich von HTML: Es vermeidet nicht nur die schlechten Konzepte von HTML, sondern hat auch eine andere Syntax als HTML. Zudem ist XML Unicode-fähig: Tags, Attribute und Inhalte können in jeder Zeichenkettenkodierungen enthalten sein, die von Unicode definiert wird.

Sehen wir uns einen kleinen Ausschnitt aus dem Quellcode dieses Buchs an:

```
<title>Cutting-Edge Applications</title>
<abstract>
    <para>
        If you realize that all things change,
1         there is nothing you will try to hold on to.
    </para>
</abstract>
```

In diesem Beispiel sehen Sie die Verwendung von Tags, mit denen eine Strukturierung und logische Auszeichnung vorgenommen wird. Im Gegensatz zu HTML

- unterscheiden Tags Klein- und Großschreibung,
- sind Leerräume bedeutend,

- muss jedes öffnende Tag stets ein entsprechendes schließendes Tag haben oder selbst schließend sein (z.B. <xref/>),
- können Dokumente eine beliebige gültige Document Type Definition haben.

Um es zusammenzufassen: XML reduziert die Komplexität von SGML, enthält aber trotzdem alle notwendigen Funktionen für die strukturelle Auszeichnung, einschließlich der Definition von benutzerspezifischen Dokumententypen.

Vorteile von XML

Aber warum XML? Bei all diesen formalen Definitionen, Datenblättern und Kurzübersichten erkennen Entwickler manchmal nicht auf den ersten Blick den Nutzen für ihre tägliche Arbeit. Warum sollten Sie jetzt XML an Stelle von Word oder Notes verwenden? Oder Ihrem eigenen proprietären Speicherformat? Oder einer relationalen Datenbank?

Gegen proprietäre Formate spricht hauptsächlich ein Argument: Sie sind proprietär. Daten, die auf einem heterogenen Netzwerk, wie etwa dem Internet, verwendet werden sollen, müssen für alle Typen von Computern benutzbar sein, die mit diesem verbunden sind. XML besteht aus reinem Text (im Gegensatz zum binären Format der meisten proprietären Anwendungen), wodurch es auf allen derzeitigen Computerplattformen unterstützt werden kann. Übrigens kommen proprietäre Formate (z.B. in öffentlichen Dokumenten) gar nicht erst in Betracht. Oder möchten Sie sich auf die Gnade eines einzelnen Herstellers verlassen, der sein Format nach Gutdünken ändern kann oder es sogar ganz abschafft? XML ist lizenzfrei, herstellerneutral und plattformunabhängig.

XML bietet Mittel zur Strukturierung des Inhalts, unterscheidet sich aber von relationalen Datenbanksystemen. Es bietet kein relationales Modell. Vielmehr ermöglicht es, eine unbegrenzte Anzahl von Verschachtelungen, was ein Datenbanksystem nicht bewältigt. Auf der anderen Seite fehlen ihm einige Funktionen, die ein RDBMS bietet, wie etwa stringente Feldtypen, Randbedingungen, Schlüssel usw. Natürlich gibt es auch Ähnlichkeiten zwischen den beiden Konzepten, und es gibt sogar Bemühungen, eine SQL-ähnliche Abfragesprache für XML-Dokumente zu generieren. Über den Erfolg vom XML sollten Sie jedoch nicht vergessen, dass herkömmliche RDBMS durchaus nützlich sind. Sie bieten wichtige Verarbeitungsfunktionen, die in XML kaum modelliert werden können, und sie sind von Anfang an auf Schnelligkeit ausgerichtet.

Der große und alles in den Schatten stellende Vorteil von XML ist die Trennung der logischen Struktur vom Layout. XML-Dokumente können in jede beliebige Darstellungsform umgewandelt werden: HTML, PostScript, PDF,

RTF, Klartext, Audio, Braille – alles aus einer einzigen Quelle. Da Sie XML (Klartextdokumente mit Ihrer bevorzugten Skriptsprache analysieren können, wird es Ihnen nicht schwer fallen, Hyperlinks dynamisch zu ändern, den Inhalt von Elementen zu ändern oder Strukturen einer Datenbank zuzuordnen.

Wenn Sie immer noch nicht überzeugt sind, schauen Sie sich einmal die Document Type Definitions an, die entwickelt werden oder bereits in Gebrauch sind. XML ist eher die Technik im Hintergrund, die Würze gibt die Anwendung, welche XML verwendet.

Wofür wird XML verwendet?

Als Auszeichnungssprache für strukturierte Informationen wird XML natürlich in Content Management-Systemen, Archivierungsprogrammen und Firmen-Dokumentarchiven eingesetzt. Es gibt jedoch eine Vielzahl andere XML-Anwendungen und Unterprotokolle. Aufgrund der Offenheit dieses Standards entstanden in kurzer Zeit viele DTDs.

7.2.2 DocBook

Die beliebte DocBookX-DTD besteht aus einer Reihe von Tags zur Beschreibung von Büchern, Atikeln und anderen Prosadokumenten, insbesondere technischen Dokumentationen. Sie wurde ursprünglich 1991 vom Verlag O'Reilly als SGML-DTD für den eigenen Gebrauch entwickelt und errang schnell Beliebtheit bei Autoren, so dass sie bald auch in anderen Verlagshäusern eingesetzt wurde. O'Reilly unterstützte dies, indem sie die DTD zur weiteren Entwicklung an die Davenport-Gruppe übergab. Mitte 1998 übernahm OASIS (Organization for the Advance of Structured Information Standards) offiziell die Pflege von DocBook. Als XML zunehmend populärer wurde, schrieb Norman Walsh eine inoffizielle XML-Version (3.1). Derzeit gibt es Bemühungen, diese in eine offizielle Version umzuwandeln. Wahrscheinlich kommt DocBook 5 sowohl als SGML- als auch als XML-Version heraus.

Als wir begannen, dieses Buch zu schreiben, stand für uns fest, dass wir ein offenes Format wie XML verwenden wollten. Wir entschieden uns für die DocBook-DTD, da sie alle Funktionen besitzt, die wir jemals brauchen würden. Hier gibt es alle Elemente, die typischerweise in technischen Dokumentationen benutzt werden, und selbst einige esoterische Elemente sind vorhanden – oder haben Sie jemals in Ihrem Textverarbeitungsprogramm ein Element `MouseButton` gesehen (aus der Kurzreferenz: der konventionelle Name einer Maustaste)?

XML und DocBook bieten einige klare Vorteile für uns. Wir können CVS zur Versionskontrolle sowohl für die PHP-Beispiele als auch für die Buchdateien verwenden. Die Umwandlung nach HTML ist einfach. Sie lässt sich mit PHP

oder einem Stylesheet-Prozessor wie XT von James Clark bewerkstelligen. Xmetal von SoftQuad bietet komfortable Bearbeitungsmöglichkeiten, die intuitive visuelle Bearbeitungen über Cascading Style Sheets (CSS) ermöglichen und sich zur Anzeige in Autorensystemen eignen (siehe hierzu Abbildung 7.3).

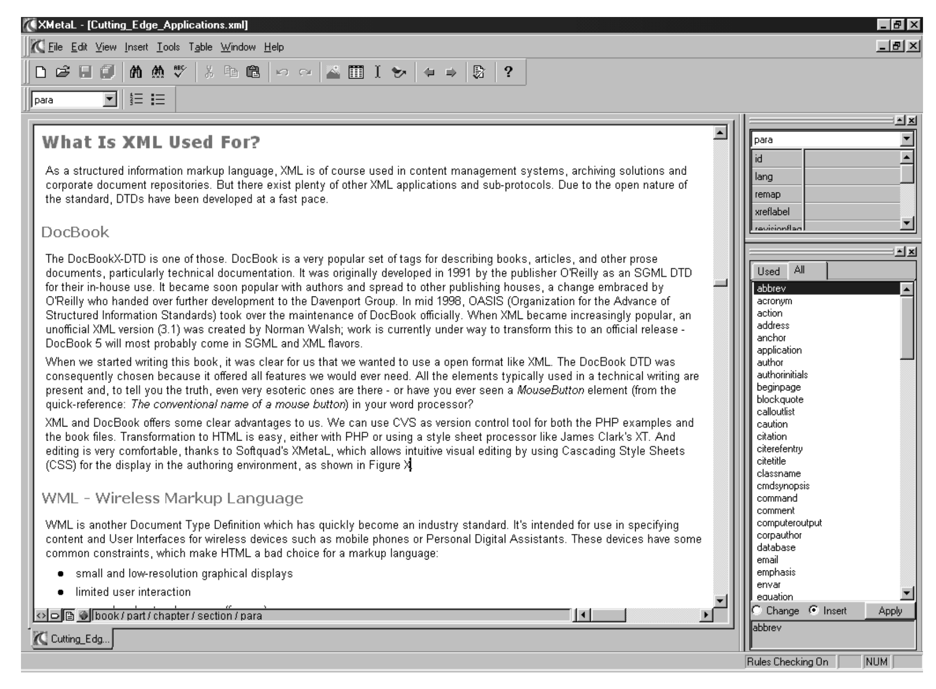

Abbildung 7.3: XMetal von SoftQuad – ein XML-Autorensystem, mit welchem dieses Buch erstellt wurde

7.2.3 WML – Wireless Markup Language

Eine weitere Document Type Definition, die schnell zum Industriestandard wurde, ist WML. Sie ist für die Verwendung in spezifischen Inhalten und Benutzerschnittstellen für drahtlose Geräte, wie Mobiltelefone oder Personal Digital Assistants, gedacht. Aufgrund der Beschränkungen, die diese Geräte haben, eignet sich HTML hier nicht so gut als Auszeichnungssprache:

- Sie haben kleine und niedrig auflösende grafische Anzeigen.
- Sie ermöglichen nur begrenzt Benutzerinteraktionen.
- Sie ermöglichen (bisher) nur einen Schmalbandzugang für Netzwerke.
- Sie haben nur eine begrenzte Rechnerkapazität.

Diese Probleme spricht WML an. Sie unterteilt den Inhalt in kleine Teile (»Karten«) und fasst sie in größere Informationseinheiten (»Decks«) zusammen. Um permanenten Netzwerkzugriff zu vermeiden, definiert WML in XML eine Reihe von Skriptprozeduren für die Clientseite, z.B. um Variablen auf dem Clientrechner zu setzen und auf diese zuzugreifen. Aufgrund des begrenzten Platzes auf dem Bildschirm ist es recht schwierig, sinnvolle Navigationspfade auf portablen Geräten zu generieren. WML zwingt den Benutzeragenten, den WAP-Browser, den Navigationsverlauf aufzuzeichnen. WML-Dokumente können diesen benutzen, welches den Autor entlastet, indem er einige Verantwortung auf den Benutzeragenten überträgt.

7.2.4 RDF – Resource Description Framework

Die RDF-Spezifikation definiert eine Sprache, um Metadaten über die Webressource im XML-Format zu speichern. Das Web mit seinen Millionen HTML-Seiten ist für automatische Maschinen, wie Spiders oder Roboter, sehr schwierig zu verarbeiten. Suchmaschinen kommen jeden Tag an ihre Grenzen, und selbst die besten Algorithmen können keine sinnvollen Suchergebnisse garantieren, wie jeder weiß, der das Web für die professionelle Forschung verwendet. Auf Webseiten kann nur eine Volltextsuche durchgeführt werden, was eine sehr begrenzte Suchmethode ist.

Die aktuellen HTML-Versionen ermöglichen eine einfache Speicherung von Metadaten zu einem Dokument. Wie Sie vielleicht wissen, können Sie Metatags verwenden, um Schlüsselwörter für ein Dokument, eine kurze Zusammenfassung oder Informationen über den Autor zu definieren. Aber was ist, wenn Sie die Erstellungsdaten des Dokuments speichern wollen? Oder Informationen über die Editoren? Kein Bibliograph wird die Metatags von HTML ernst nehmen.

1998 bildete das W3C ein Komitee, um ein Format zur Definition von Metadaten zu entwickeln, und verabschiedete am 22.Februar 1999 das Resource Description Framework (RDF).

RDF ist eine Erweiterung des Formats, das ursprünglich für PICS verwendet wurde, einem Inhaltsbewertungssystem. Es ersetzt zunehmend die Dublin Core Metadata für den Resource-Discovery-Standard, ein anderes Verfahren zur Klassifizierung von Metadaten. RDF hat sich rasch als Standardmethode für den globalen Austausch von Metadaten im Internet durchgesetzt.

7.2.5 XML-Dokumente

XML-Dokumente bestehen aus der Auszeichnung und dem Inhalt (den sogenannten Zeichendaten in der XML-Sprache) im Unicode-Zeichensatz. Es gibt verschiedene Arten der Auszeichnung, auf die wir in den folgenden Abschnitten ausführlich eingehen.

Elemente

Jeder, der bereits mit HTML gearbeitet hat, wird mit Elementen vertraut sein. Sie kennzeichnen die Bedeutung eines bestimmten inhaltlichen Abschnitts. Für jedes Element wird in XML ein schließendes Tag benötigt, (z.B. für das HTML-Tag `<img>`). Es gibt jedoch eine gesonderte Schreibweise zur Kennzeichnung von leeren Tags.

```
<xref   linkend="end"/>
```

Bedenken Sie, dass die Verschachtelung von Tags signifikant ist. Falsch verschachtelte Tags führen zu schlecht geformten Dokumenten.

Attribute

Jedes Element kann Attribute haben. Attribute sind Namen / Wert-Paare, die in den Tags hinter dem Elementnamen auftauchen und die Eigenschaft eines Elements angeben. Attributwerte müssen in Anführungsstrichen angegeben werden. Jeder Attributname darf in einem Tag nur einmal vorkommen.

XML-Dokumente können optional (und ungeachtet der Dokumententypdefinition) zwei Standardattribute besitzen: `xml:lang` und `xml:space`. Das Attribut `xml:lang` wurde definiert, weil Sprachunabhängigkeit eines der wichtigsten Ziele von XML ist.

Ohne jedoch zu wissen, in welcher Sprache ein Text geschrieben ist, kann eine Anwendung einen Text nicht anzeigen, auf Rechtschreibung überprüfen oder indizieren. Die Unicode-Unterstützung für XML ist nicht sehr hilfreich, wenn der Autor einem bestimmten Teil eines Dokumentes kein Sprachtag zuweisen kann. Aus diesem Grund wurde das Attribut `xml:lang` eingeführt.

```
<p>Worldwide declarations of love</p>
<p xml:lang="It">Ti amo.</p>
<p xml:lang="De">Ich liebe Dich.</p>
<p xml:lang="X-Klingon">qabang</p>
```

Ein Sprachbezeichner kann folgendermaßen aussehen:

- ein zweibuchstabiger Sprachcode nach ISO 639
- ein Sprachcode, der bei der Internet Assigned Numbers Authority (IANA) registriert wurde; er beginnt mit dem Präfix »i-« (oder »I-«)
- ein benutzerdefinierter Code, der mit dem Präfix »x-« (oder »X-«) beginnt

Das andere Standardattibut `xml:space` ist nicht so einfach zu verstehen und zu verwenden. Wie bereits erwähnt, ist Leerraum in XML signifikant. Er wird an die verarbeitende Anwendung übergeben. Wenn Sie unsere Kodierstil-Richtlinien gelesen haben, wissen Sie, dass Leerraum wichtig ist, um den Programmcode zu strukturieren und Leerzeilen einzufügen, welche die Lesbarkeit verbessern. So wird das Attribut zwar für die strukturelle Auszeichnung

eingesetzt, hat aber keine Auswirkung auf die Auszeichnung selbst oder die Zeichendaten. Es bietet dem Autor die Möglichkeit, den Leerraum, wenn gewünscht, zu erhalten.

Mit der Einführung von `xml:space` reagierte das XML-Komitee auf die verschiedenen Ansichten zum Thema Leerraum. Das Attribut kann zwei Werte haben: `preserve` oder `default`. Bei jedem Element, welches das Attribut `xml:space="preserve"` enthält, wird Leerraum als signifikant betrachtet und an die verarbeitende Anwendung unverändert übergeben. Der Wert `default` teilt der Anwendung mit, dass die Standardverarbeitung angewendet werden soll. Beide Standardattribute werden auf die Unterelemente vererbt, solange sie nicht explizit in einem Element zurückgesetzt werden.

> **Hinweis:** Ein XML-Prozessor ist ein Programm, das zum Einlesen von XML-Dokumenten verwendet wird. Der XML-Prozessor ermöglicht einer Anwendung, auf die Struktur und den Inhalt eines XML-Dokuments zuzugreifen. In diesem Buch verwenden wir die Begriffe *XML-Prozessor* und *XML-Parser* für dieselbe Art von Software.

Verarbeitungsbefehle

Ein weiterer »Element«-Typ, den Sie in XML-Dokumenten finden, ist der Verarbeitungsbefehl. Er wird verwendet, um Teile in einem Dokument zu definieren, die nicht vom regulären Parserprogramm interpretiert werden sollen, sondern von einer speziellen Verarbeitungsroutine. Er setzt sich zusammen aus <? und einem Zielnamen, der die Anwendung angibt, auf welche sich der Befehl richtet. Das lange PHP-Tag (`<?php`) ist ein Beispiel hierfür; es wird in XML-Dokumenten verwendet, um PHP-Code zu markieren.

> **Hinweis:** Um XML-kompatibel zu sein, müssen Sie den Befehl `short_tags` in Ihrer PHP-Konfiguration auf `Off` setzen, und konsequent das lange öffnende Tag `<?php` verwenden. Das kurze öffnende Tag würde XML verwirren, da es kein gültiger Verarbeitungsbefehl wäre. Auf der anderen Seite würden sich Tags, wie `<?xml`, nicht mit PHP vertragen, da XML von PHP für Programmcode gehalten und dementsprechend ein Analysefehler angezeigt würde.

Entitäten

Der Teil des Textes, der keine Auszeichnung darstellt, wird als Zeichendaten des Dokuments bezeichnet. Innerhalb dieses Textes muss ein Autor Sonderzeichen, wie < oder >, einfügen können, die normalerweise den Beginn oder das Ende von Auszeichnungsabschnitten kennzeichnen. Genau wie HTML kennt XML die Schreibweise von Entitären. Es gibt fünf vordefinierte Entitäten:

Entität	Zeichen	Symbol
<	<	kleiner als
>	>	größer als
&	&	Ampersand
"	"	doppelte Anführungsstriche
'	'	einfache Anführungsstriche (Apostroph)

Hinweis: Wenn Sie eine Document Type Definition benutzen, müssen diese Entitäten deklariert werden, wenn Sie sie verwenden wollen.

Mithilfe von Zeichenreferenzen können Sie ein beliebiges Unicode-Zeichen in Ihr Dokument einfügen. Sie werden wie andere Referenzen geschrieben, wobei hinter dem Ampersand ein Pfund-Zeichen (#) steht. Danach wird entweder ein dezimaler oder hexadezimaler Verweis auf die Unicode-Position eingefügt. Zum Beispiel referieren sowohl &#8478 als auch &#x221E auf das Unendlich-Zeichen (8). Entitäten sind jedoch nicht auf ein einzelnes Zeichen begrenzt. Sie können beliebig lang sein. Eine DTD könnte beispielsweise eine Entität &footer definieren, die den Inhalt "Copyright (c) 2000 Pearson Education" enthält.

Kommentare

XML verwendet dieselbe Schreibweise für Kommentare wie HTML: <!--comment-->. Kommentare können beliebige Daten außer der Zeichenfolge – - enthalten, und zwischen den Auszeichnungseinträgen an einer beliebigen Stelle in Ihrem Dokument eingefügt werden. In der XML-Spezifikation ist ausdrücklich angegeben, dass Kommentare nicht Teil des Dokumenteninhalts sind. Es wird kein Parser benötigt, um sie an die verarbeitende Anwendung zu übergeben. Sie können Kommentare also nicht für versteckte Anweisungen o. ä. verwenden, wie Sie es möglicherweise von HTML gewohnt sind (denken Sie an die Verwendung der Kommentar-Tags, mit denen Sie Java-Script vor älteren Browsern verstecken).

CDATA-Abschnitte

Ein spezieller Inhaltstyp sind CDATA-Abschnitte. Sobald Sie größere Abschnitte von Programmcode in Ihr XML-Dokument einfügen möchten (die viele < und & enthalten), werden Sie die Standardmethode, einzelne Zeichen per Entität zu referenzieren, als sehr lästig empfinden. HTML bietet hierfür das Tag <pre>, mit welchen die Auszeichnungsinterpretation für einen Abschnitt ausgeschaltet wird. Da XML solche integrierten Tags fehlen, können wir dies nicht machen. Um diesen Mangel zu beseitigen, können Sie Abschnitte in XML wie folgt als CDATA markieren:

```
<![CDATA [
   print("<a href="script.php3?foo=bar&baz=foobar");
]]>
```

Innerhalb eines CDATA-Abschnitts können alle Zeichen, mit Ausnahme der Sequenz]]> vorkommen.

Prolog des Dokuments

XML-Dokumente sollten (müssen aber nicht) mit einer XML-Deklaration beginnen, die angibt, welche XML-Version verwendet wird. Diese Versionsinformation ist Teil des Dokumentenprologs:

```
<?xml version="1.0"?>
    <greeting>Hello, world!
</greeting>
```

Dadurch, dass diese Information zu Beginn des Dokuments angegeben ist, kann ein Prozessor entscheiden, ob er die XML-Version des Dokuments bearbeiten kann. Es eignet sich auch als Methode, um den Dokumententyp zu identifizieren. So wie *#!/bin/sh* im Dateikopf angibt, dass es sich um ein Shellskript handelt, identifiziert die XML-Deklaration ein XML-Dokument.

Den zweiten wichtigen Teil des Prologs bildet die Dokumententyp-Deklaration. Verwechseln Sie diese nicht mit der Document Type Definition (DTD). Die Dokumententyp-Deklaration zeigt auf oder enthält eine DTD! Die DTD besteht aus Auszeichnungsdeklarationen, die eine »Grammatik« für XML-Dokumente zur Verfügung stellen. Die Dokumententyp-Deklaration kann entweder auf eine externe DTD verweisen, die Auszeichnungsdeklarationen direkt einbinden oder beides. Die DTD für ein Dokument besteht aus beiden Teilen. Hier ein Beispiel einer Dokumententyp-Deklaration:

```
<!DOCTYPE book SYSTEM "docbookx.dtd">
```

Die Dokumententyp-Deklaration hat den Namen `book` und zeigt auf eine externe DTD namens `docbookx.dtd`. Es sind keine DTDs eingebettet.

Wenn ein Dokument die vollständige DTD, aber keine externen Entitäten enthält, wird es als standalone-Dokument bezeichnet und als solches in der XML-Deklaration markiert:

```
<?xml version="1.0" standalone='yes'?>
```

Dies kann für einige Anwendungen nützlich sein, z.B. um Dokumente über das Netz zu schicken, wenn Sie nur einen einzelnen Dokumentendatenstrom öffnen möchten. Beachten Sie, dass Sie selbst XML-Dokumente mit externen DTDs in selbstständige Dokumente umwandeln können, indem Sie die DTD und die externen Entitäten in den Prolog des Dokuments importieren.

Dokumentenstruktur

Nun kennen Sie alle Teile, die ein XML-Dokument ausmachen: Elemente (mit Attributen), Verarbeitungsbefehle, Entitäten, Kommentare und CDATA-Abschnitte. Aber wie wird aus diesen Einzelteilen ein sinnvolles XML-Dokument?

Die XML-Spezifikation definiert nur eine sehr allgemeine Dokumentenstruktur. Sie besagt, dass jedes wohlgeformte Dokument folgende Eigenschaften hat (mehr über die Bedeutung von »wohlgeformt« später):

- kann einen Prolog haben, der die XML-Version und die verwendete DTD angibt
- muss genau ein Stammelement und kann eine beliebige Anzahl von Elementen unterhalb des Stamms enthalten
- danach kann Verschiedenes folgen

Der letzte Teil, hier etwas flapsig »Verschiedenes« bezeichnet, wird von vielen Leuten als Konzeptfehler von XML betrachtet. Er erschwert möglicherweise die Analyse der XML-Dokumente, da Sie sich nicht darauf verlassen können, dass das Dokumentenende das schließende Stammelement ist. Wenn Sie ein Dokument über eine Netzwerkverbindung analysieren, können Sie beispielsweise die Verbindung nicht einfach schließen, nachdem Sie das schließende Stammelement erhalten haben. Sie müssen warten, bis der Server die Verbindung von selbst schließt, da ja immer noch »Verschiedenes« folgen kann.

Bisher haben wir noch kein Wort über die Syntax und Struktur des Elements verloren, das für den ganzen Zauber von XML verantwortlich sein soll: die Dokumententyp-Definition. Tatsächlich ist es die DTD, die einem XML-Dokument die Bedeutung verleiht. Sie definiert seine Syntax, die Abfolge und Verschachtelung von Tags, mögliche Elementattribute, Entitäten – kurz, die gesamte Grammatik. Komplexe DTDs zu schreiben ist keine einfache Aufgabe, und darüber wurden schon ganze Bücher geschrieben. Da Sie sich als XML-Benutzer normalerweise nicht mit dieser Aufgabe befassen müssen, gehen wir auf dieses Thema nicht näher ein. Statt dessen möchten wir uns ein weiteres XML-Konzept ansehen, das für Ihre tägliche Arbeit wichtiger sein könnte.

XML-Namensräume

Sie haben verschiedene XML-Anwendungen (Document Type Definitions) gesehen und erfahren, wozu sie verwendet werden. Was passiert jedoch, wenn Sie ein einzelnes XML-Dokument erzeugen möchten, das Elemente aus zwei verschiedenen DTDs enthält? Das Element `<part>` könnte in einer DTD beispielsweise den Teil eines Buches bezeichnen, während es in einer anderen DTD den Herstellungsteil kennzeichnet. Ohne eine Möglichkeit, diese beiden Namensräume voneinander zu trennen, würden die beiden Elementnamen kollidieren.

Wie können diese unterschiedlichen Elemente identifiziert werden? Sie müssen dem Element einen Bezeichner zuordnen, z.B. `<part namespace = "book">` oder, wenn Sie Attribute vermeiden wollen, `<book:part>` und `<manufacturing:part>`.

Das W3C erfuhr früh von diesen Mängeln in XML und führte eine neue Spezifikation ein: Namensräume in XML, die als Empfehlung am 14.Januar 1999 verabschiedet wurde.

XML-Namensräume bieten die Möglichkeit, in einem XML-Dokument mehrere Namensräume zu verwenden, die jeweils über einen URI(Uniform Resource Identifier)-Bezeichner gekennzeichnet werden. Dieses Verfahren wird in der RDF(Resource Description Framework)-DTD verwendet. Betrachten Sie folgendes Beispiel aus der RDF-Spezifikation:

```
<?xml version="1.0"?>
<rdf:RDF    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:s="http://description.org/schema/">
    <rdf:Description about="http://www.w3.org/Home/Lassila">
        <s:Creator>Ora Lassila</s:Creator>
    </rdf:Description>
</rdf:RDF>
```

Hier werden zwei Namensräume definiert, einer mit dem Namen `rdf` und der andere mit dem Namen `s`. Nach der Definition wird auf einen Namensraum verwiesen, indem er als Präfix (durch Doppelpunkt getrennt) einem Elementnamen vorangestellt wird. Damit wird effektiv die Kollision verschiedener logischer Bedeutungen und syntaktischer Definitionen vermieden.

> **Hinweis:** Die URI in einem Namensraum-Bezeichner ist keine DTD. Es wäre natürlich schön, über XML-Namensräume auf verschiedene DTDs zeigen zu können, aber es gibt derzeit noch viele technische Probleme bei diesem Ansatz. Hiermit beschäftigt sich gerade das W3C in Form der XML-Schemadefinition, die sich zum Zeitpunkt der Bucherstellung noch in der Entwickelung befand.

EBNF – oder »Was ist dies nun schon wieder?«

Als Webentwickler haben Sie häufig die Aufgabe, Spezifikationen zu lesen: Projektspezifikationen, formale Sprachspezifikationen oder Whitepapers zu Standards. Beim Lesen einiger Spezifikationen vom W3C (die bekanntesten sind wahrscheinlich die HTML- und XML-Dokumente), stolpern Sie wahrscheinlich über eine seltsame Mischung von Zeichen, die vermutlich eine Grammatikdefinition bilden.

```
document ::= prolog element Misc*
```

Dies ist die allererste Syntaxdefinition in der XML-Spezifikation. Sie definiert die grundlegende Struktur eines XML-Dokuments. Die verwendete Notation wird als »erweiterte Backus-Naur-Form oder EBNF bezeichnet. Sobald Sie die Grundlagen von EBNF verstanden haben, werden Sie formale Spezifikation sehr viel einfacher verstehen.

EBNF ist eine formale Methode, um die Syntax einer Programmiersprache zu definieren, so dass hinterher eindeutig feststeht, was gültig oder zugelassen ist. Sie wird auch in vielen anderen Standards verwendet, wie etwa Protokoll- oder Datenformaten, sowie in einigen Auszeichnungssprachen, wie XML oder SGML. EBNF ermöglicht eine vollständige und eindeutige Grammatikdefinition. Deshalb gibt es einige Softwaretools, die eine Reihe von EBNF-Regeln automatisch in einen Parser umwandeln. Programme, die dies tun, werden Compiler Compiler genannt. Der bekannteste von ihnen ist YACC (Yet Another Compiler Compiler), aber es gibt noch eine Reihe weiterer.

Sie können EBNF als eine Reihe von Regeln, den so genannten Produktionen oder Produktionsregeln, betrachten. Jede Regel beschreibt einen Teil der gesamten Syntax. Sie beginnen bei einem Startsymbol (traditionell mit S bezeichnet) und definieren anschließend Regeln dafür, wodurch Sie dieses Symbol ersetzen können. Schritt für Schritt entsteht so eine komplexe Sprachgrammatik, die sich aus den Zeichenfolgen zusammensetzen, die Sie produzieren können, wenn Sie diese Regeln befolgen.

Wenn Sie sich noch einmal das Beispiel oben betrachten, stellen Sie fest, dass es sich um eine Zuordnung handelt. Es gibt ein Symbol auf der linken Seite, einen Zuordnungsoperator (der auch in Form von := geschrieben werden kann) und eine Liste der Werte auf der rechten Seite. Sie arbeiten die Zuordnungen so lange ab, bis es auf der linken Seite keine Symboldefinitionen mehr gibt. Danach sollte auf der rechten Seite der Zuordnung kein Symbol mehr stehen, sondern nur noch die endgültige Zeichenkette, die ein atomarer Wert ist.

EBNF definiert drei Operatoren, die Sie möglicherweise schon aus regulären Ausdrücken kennen:

Operator	Bedeutung
?	optional
+	muss einmal oder mehrmals erscheinen
*	muss null-mal oder mehrmals erscheinen

Zur Definition der Sprachgrammatik mit der Sie Fließkommazahlen verwenden können, sähe die EBNF-Notation folgendermaßen aus:

```
S := SIGN? D+ (. D+)?
D := [0-9]
SIGN := "+"|"-"
```

Die erste Zeile definiert das Startsymbol mit den folgenden Sequenz:

- ein optionales Vorzeichen, entweder + oder -
- ein oder mehrere Elemente der D-Produktion
- optional ein Punkt, und wieder ein oder mehrere Elemente der D-Produktion

Beachten Sie, dass Operatoren in EBNF auch für Symbolgruppen verwendet werden können: `(. D+)?` bedeutet, dass dieser Ausdruck optional ist.

Die zweite Zeile liest zunächst die Endwerte (Atome) der D-Produktion, in diesem Fall die Ziffern Null bis Neun. Die Syntax entspricht jener für reguläre Ausdrücke. Ein Bereich wird in eckigen Klammern angegeben.

Die dritte Zeile definiert die beiden möglichen Zeichen. Das Zeichen `A|B` kennzeichnet Alternativen »A oder B aber nicht beides«.

Dies ist eine sehr einfache Erklärung von EBNF. Die XML-Spezifikation enthält weitere Definitionen zur Syntax, z.B. Randbedingungen für Gültigkeit und Wohlgeformtheit. Betrachten Sie hierzu den Abschnitt »Notation« der Spezifikation. Deshalb gehen wir hier nicht näher darauf ein. Weitere Informationen zu EBNF finden Sie in jedem modernen Compilerbuch.

Gültigkeit und Wohlgeformtheit

Es gibt zwei Arten von XML-Dokumenten, die bestimmte Bedingungen erfüllen: gültige Dokumente und wohlgeformte Dokumente. Ein XML-Dokument ist wohlgeformt, wenn es den grundlegenden Syntaxrichtlinien von XML entspricht:

- Es enthält ein Stammelement und eine beliebige Anzahl von Elementen unterhalb dieses Elements.
- Elemente sind korrekt verschachtelt.
- Attribute tauchen nur einmal pro Element auf und sind in einfache oder doppelte Anführungszeichen eingeschlossen. Sie enthalten keine direkten oder indirekten Entitätenverweise auf externe Entitäten. Sie enthalten auch kein öffnendes Tag (<).
- Mit Ausnahme der Standardentitäten müssen alle Entitäten vor ihrer ersten Verwendung deklariert werden.
- Entititäten verweisen nicht rekursiv auf sich selbst.

Hier ein Beispiel für ein wohlgeformtes XML-Dokument:

```
<greeting>Hello world.</greeting>
```

Es ist jedoch kein gültige Dokument. In der XML-Spezifikation ist dies folgendermaßen definiert: Ein XML-Dokument ist gültig, wenn ihm eine Dokumententyp-Deklaration zugeordnet ist und wenn das Dokument den Randbedingungen entspricht, die darin genannt sind. Dies bedeutet, das jedes gültige XML-Dokument auch wohlgeformt ist. Ein wohlgeformtes Dokument kann ungültig sein, wenn es nicht der Syntax entspricht, die in der zugehörigen DTD definiert wurde. Ein nicht wohlgeformtes Dokument kann niemals gültig sein. Ein nicht wohlgeformtes Dokument ist kein XML-Dokument. Es enthält schwerwiegende Fehler, und XML-Parser werden an dieser Stelle angewiesen, die Verarbeitung abzubrechen. Die Unterscheidung zwischen gültig und wohlgeformt hat für XML zwei wichtige Bedeutungen: Zunächst führt es zu zwei Klassen von XML-Parsern, jene, für welche die Gültigkeit eines XML-Dokuments zählt, und jene, für die dies nicht wichtig ist: Parser mit Gültigkeitsüberprüfung und Parser ohne Gültigkeitsüberprüfung. Ein Designziel der XML-Spezifikation ist die einfache Anwendbarkeit für den Programmierer. Für einen durchschnittlichen Programmierer ist es tatsächlich recht einfach, einen Parser ohne Gültigkeitsprüfung zu schreiben. Einen Parser mit Gültigkeitsprüfung zu schreiben, ist dagegen eine ganz andere Sache.

Die Unterscheidung zwischen gültig und wohlgeformt unterteilt XML-Anwendungen außerdem in zwei Kategorien. Eine Gruppe von Anwendungen betrachtet XML als erweitertes Datenspeicherformat. Wohlgeformte Dokumente werden zur Datenspeicherung und Anzeige verwendet. Für diese Aufgabe ist eine DTD nicht notwendig, ein wohlgeformtes Dokument reicht aus. Mit diesem Ansatz könnten Sie Programmcodes bis zu einem gewissen Grad wiederverwenden, z.B. könnten Sie den Code zur Analyse der Daten und zur Erzeugung der Tags in späteren Anwendungen wiederverwenden. Sobald Sie jedoch Informationen als Informationen austauschen wollen (und sie nicht mehr als reine Daten behandeln), müssen Sie dem Dokument eine Bedeutung geben und es mit einer DTD verknüpfen. Für Anwendungen, die sich mit der Verarbeitung und dem Austausch von Informationen befassen, sind nur gültige Dokumente zulässig.

Nachdem Sie nun die Grundlagen von XML und verwandte Themen kennen gelernt haben, wollen wir die gewonnenen Erkenntnisse in die Praxis umsetzen, indem wir uns Expat ansehen, einen Parser ohne Gültigkeitsprüfung, der in PHP integriert ist.

7.2.6 PHP und Expat

Expat ist der Parser, der für die XML-Verarbeitung in Mozilla, Apache, Perl und vielen anderen Projekten zuständig ist. Er kann seit der Version 3.0.6 in PHP eingebunden werden und ist Teil der offiziellen Apache-Distribution seit der Apache-Version 1.3.9. Da Expat ein Parser ohne Gültigkeitsprüfung ist, ist es schnell und klein – und eignet sich damit gut für Webanwendungen.

Ereignisbasierte API

Es gibt zwei Arten von APIs für XML-Parser: Parser, die auf Baumstrukturen basieren und eine Schnittstelle für das Document Object Model (mehr dazu später) bereitstellen, und jene, die XML-Dokumente mit einem ereignisbasierten Ansatz verarbeiten. Expat stellt eine ereignisbasierte API zur Verfügung.

Ereignisbasierte Parser nehmen eine datenbezogene Sichtweise der XML-Dokumente ein. Sie analysieren das Dokument von oben nach unten und melden Ereignisse, wie etwa den Beginn eines Elements, das Ende eines Elements, den Beginn von Zeichendaten usw. an die Anwendung, in der Regel über eine Rückruffunktion. Das Beispieldokument »Hello World« weiter oben wird von einem ereignisbasierten Parser als eine Reihe dieser Ereignisse berichtet.

1. Element öffnen: greeting
2. CDATA-Abschnitt öffnen, Wert: Hello World
3. Element schließen: greeting

Im Gegensatz zu baumstrukturbasierten Parsern, erzeugen sie keine Strukturdarstellung des Dokuments. Dies ermöglicht einen low-level-Zugriff und ist in Bezug auf Geschwindigkeit und Ressourcennutzung sehr viel effizienter. Es muss nicht das gesamte Dokument im Speicher geladen sein. Die Dokumente können sehr viel größer sein als Ihr Systemspeicher. Trotzdem haben Sie immer noch die Möglichkeit, wenn nötig, eine reine Baumstruktur zu erstellen. Bevor ein Dokument analysiert werden kann,. fordert Sie ein ereignisbasierter Parser in der Regel auf, Rückruffunktionen zu registrieren, die aufgerufen werden, wenn bestimmte Ereignisse eintreten. Expat bildet hier keine Ausnahme. Er definiert sechs mögliche Ereignisse und eine Standardroutine:

Ziel	Funktion	Beschreibung
Elemente	`xml_set_element_handler()`	Öffnen und Schließen von Elementen
Zeichendaten	`xml_set_character_data_handler()`	Anfang von Zeichendaten
Externe Entitäten	`xml_set_external_entity_ref_handler()`	Auftreten einer externen Entität
Nicht analysierte externe Entitäten	`xml_set_unparsed_entity_decl_handler`	Auftreten einer nicht analysierten externen Entität
Verarbeitungsbefehle	`xml_set_processing_instruction_handler()`	Auftreten eines Verarbeitungsbefehls

Ziel	Funktion	Beschreibung
Notationsdeklarationen	`xml_set_notation_decl_handler()`	Auftreten einer Notationsdeklaration
Standard	`xml_set_default_handler()`	Alle Ereignisse, denen keine Bearbeitungsroutine zugeordnet ist

Beginnen wir mit einem wirklich einfachen Beispiel. Der Quellcode in Listing 7.2 stellt ein Programm dar, mit dem alle Kommentare aus einem XML-Dokument entfernt werden. Zur Erinnerung: Kommentare haben die Form `<!-- ... -->`. Das Beispiel registriert nur eine Bearbeitungsroutine, die während der Analyse für alle Ereignisse aufgerufen wird. Wenn Sie eine andere Bearbeitungsroutine registrieren, z.B. durch Verwendung von `xml_set_character_data_handler()`, würde die Standardroutine für dieses spezielle Ereignis nicht aufgerufen. Die Standardroutine verarbeitet nur »freie« Ereignisse, denen keine Routine zugeordnet ist.

```
require("xml.php3");

function default_handler($p, $data)
{
    global $count;  // count of comments found

    // Check if the current contains a comment
    if (ereg("!--", $data, $matches))
    {
       $line = xml_get_current_line_number($p);

        // Insert a tab before new lines
       $data = str_replace("\n", "\n\t", $data);

        // Output line number and comment
       print "$line:\t$data\n";

        // Increase count of comments found
       $count++;
    }

}

// Process the file passed as first argument to the script
$file = $argv[1];

$count = 0;
```

```
// Create the XML parser
$parser = xml_parser_create();

// Set the default handler for all events
xml_set_default_handler($parser, "default_handler");

// Parse file and check the return code
$ret = xml_parse_from_file($parser, $file);
if(!$ret)
{
    // Print error message and die
    die(sprintf("XML error: %s at line %d",
                    xml_error_string(xml_get_error_code($parser)),
                    xml_get_current_line_number($parser)));
}

// Free the parser instance
xml_parser_free($parser);
```

Listing 7.2: Entfernen von Kommentaren aus einem XML-Dokument

Das Beispiel funktioniert auf recht einfache Weise. Zunächst wird mit `xml_parser_create()` eine XML-Parserinstanz erzeugt. In allen folgenden Funktionen verwenden Sie den auf diese Weise erzeugten Parserbezeichner, ähnlich wie wir es bei dem Ergebnisbezeichner in den MySQL-Funktionen tun. Anschließend wird die Standardbearbeitungsroutine registriert und die Datei analysiert. `xml_parse_from_file()` ist eine anwendungsspezifische Funktion, die wir in einer Bibliothek bereitstellen. Diese Funktion öffnet die im Argument angegebene Datei und analysiert diese in Blöcken von je 4KB. Die ursprünglichen XML-Funktionen von PHP `xml_parse()` und `xml_parse_into_struct()` arbeiten nur mit Zeichenketten. Wenn Sie dagegen eigene Funktionen und Routinen zum Öffnen, Lesen und Schließen einer Datei verwenden und die Inhalte an die entsprechende Funktion übergeben, können Sie Zeit und Programmcode sparen.

Die Standardroutine überprüft, ob der vorliegende Datenabschnitt ein Kommentar ist und gibt diesen zurück, wenn es der Fall ist. Zusammen mit dem Kommentar wird auch die aktuelle Zeilennummer ausgegeben (ausgelesen mit `xml_get_current_line_number()`).

Dieses Beispiel zeigt zwar die grundlegenden Konzepte für den Aufruf des XML-Parsers, Registrierung von Rückruffunktionen und die Verarbeitung von Daten, ist aber keine exakte Wiedergabe der üblichen Verwendung eines XML-Parsers. Informationen werden nicht verarbeitet, Rohdaten werden einfach eingelesen und Zeichenketten gescannt. Nichts, was nicht auch mit einem herkömmliche regulären Ausdruck bewerkstelligt werden könnte. In

den meisten Fällen, in denen Sie XML einsetzen, möchten Sie zumindest eine einfache Darstellung der Dokumentenstruktur erhalten.

Stapel, Tiefen und Listen

Unser zweites Beispiel zeigt, wie die Elementtiefe erfasst wird, die der Parser gerade bearbeitet. In der Startelementroutine wird die globale Variable `$depth` um vier erhöht. In der Stoppelementroutine wird sie um denselben Betrag reduziert. Dies ist der kleinste Fall eines Parserstapels. Außer der Tiefenangabe werden keine Strukturinformationen gespeichert. Als XML-Pretty Printer verwendet das Beispiel die Tiefe für den richtigen Einzug des Codes. Die Funktionen der Routine wenden einfach ein Cascading Style Sheet auf die aktuellen Daten an, um eine hübsch formatierte Ausgabe zu erzeugen. Der einzige andere bemerkenswerte Teil des Codes ist folgende Zeile:

```
xml_parser_set_option($parser, XML_OPTION_CASE_FOLDING, 0);
```

Dieser Befehl deaktiviert die Großschreibung, d. .h. der Parser wird angewiesen, die Groß-/Kleinschreibung der Elementnamen beizubehalten. Wenn diese Option aktiviert ist, werden alle Elementnamen in Großbuchstaben umgewandelt. In der Regel werden Sie diese Funktion nicht wollen, da in XML die Schreibweise für Elementnamen von Bedeutung ist.

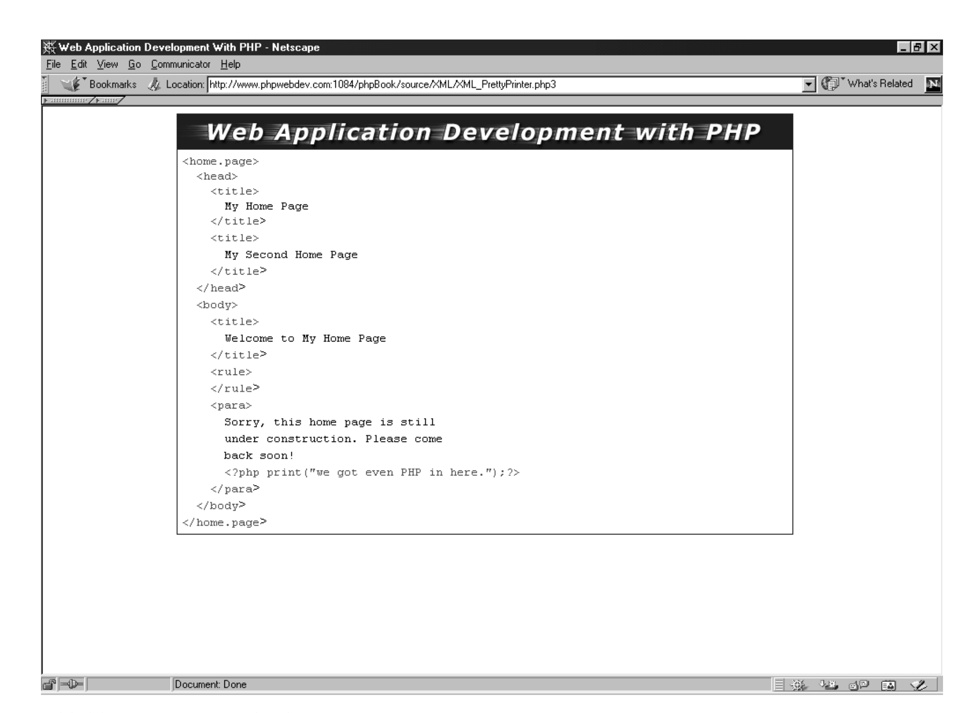

Abbildung 7.4: Ausgabe des XML-Pretty Printer

Der Einfachheit halber drucken wir den Quellcode für dieses Beispiel hier nicht ab. Sie finden ihn auf der CD-ROM. Abbildung 7.4 zeigt einen Screenshot dieser Ausgabe.

Normalerweise reicht die Verwendung einer einzigen Tiefenvariable nicht aus. Die Verwendung von ereignisbasierten Parsern führt schnell dazu, dass Sie am Ende doch Ihre eigenen Stapel oder Listen verwenden, um Informationen über die Dokumentenstruktur zu speichern. Dies zeigt auch das nächste Beispiel, das Sie in Listing 7.3 sehen.

```
require("xml.php3");

// The first argument is the file to process
$file = $argv[1];

// Initialize variables
$elements = $stack = array();
$total_elements = $total_chars = 0;

// The base class for an element
class element
{
    var $count = 0;
    var $chars = 0;
    var $parents = array();
    var $childs = array();
}

// Utility function to print a message in a box
function print_box($title, $value)
{
    printf("\n+%'-60s+\n", "");
    printf("|%20s", "$title:");
    printf("%14s", $value);
    printf("%26s|\n", "");
    printf("+%'-60s+\n", "");
}

// Utility function to print a line
function print_line($title, $value)
{
    printf("%20s", "$title:");
    printf("%15s\n", $value);
}

// Sort function for usasort()
function my_sort($a, $b)
```

```
{
    return(is_object($a) && is_object($b) ? $b->count - $a->count: 0);
}

function start_element($parser, $name, $attrs)
{
    global $elements, $stack;

    // Does this element already exist in the global $elements array?
    if(!isset($elements[$name]))
    {
        // No - add a new instance of class element
        $element = new element;
        $elements[$name] = $element;
    }

    // Increase this element's count
    $elements[$name]->count++;

    // Is there a parent element?
    if(isset($stack[count($stack)-1]))
    {
        // Yes - set $last_element to the parent
        $last_element = $stack[count($stack)-1];

        // If there is no entry for the parent element in the current
        // element's parents array, initialize it to 0
        if(!isset($elements[$name]->parents[$last_element]))
        {
            $elements[$name]->parents[$last_element] = 0;
        }

        // Increase the count for this element's parent
        $elements[$name]->parents[$last_element]++;

        // If there is no entry for this element in the parent's
        // elements' child array, initialize it to 0
        if(!isset($elements[$last_element]->childs[$name]))
        {
            $elements[$last_element]->childs[$name] = 0;
        }

        // Increase the count for this element parent in the parent's
        // childs array
        $elements[$last_element]->childs[$name]++;
    }
```

```
    // Add current element to the stack
    array_push($stack, $name);
}

function stop_element($parser, $name)
{
    global $stack;

    // Remove last element from the stack
    array_pop($stack);
}

function char_data($parser, $data)
{
    global $elements, $stack, $depth;

    // Increase character count for the current element
    $elements[$stack[count($stack)-1]]->chars += strlen(trim($data));
}

// Create Expat parser
$parser = xml_parser_create();

// Set handler functions
xml_set_element_handler($parser, "start_element", "stop_element");
xml_set_character_data_handler($parser, "char_data");
xml_parser_set_option($parser, XML_OPTION_CASE_FOLDING, 0);

// Parse the file
$ret = xml_parse_from_file($parser, $file);
if(!$ret)
{
    die(sprintf("XML error: %s at line %d",
                xml_error_string(xml_get_error_code($parser)),
                xml_get_current_line_number($parser)));
}

// Free parser
xml_parser_free($parser);

// Free helper elements
unset($elements["current_element"]);
unset($elements["last_element"]);

// Sort $elements array by element count
uasort($elements, "my_sort");
```

```
// Loop through all elements collected in $elements
while(list($name, $element) = each($elements))
{
    print_box("Element name", $name);

    print_line("Element count", $element->count);
    print_line("Character count", $element->chars);

    printf("\n%20s\n", "* Parent elements");

    // Loop through the parents of this element, output them
    while(list($key, $value) = each($element->parents))
    {
        print_line($key, $value);
    }
    if(count($element->parents) == 0)
    {
        printf("%35s\n", "[root element]");
    }

    // Loop through the childs of this element, output them
    printf("\n%20s\n", "* Child elements");
    while(list($key, $value) = each($element->childs))
    {
        print_line($key, $value);
    }
    if(count($element->childs) == 0)
    {
        printf("%35s\n", "[no childs]");
    }

    $total_elements += $element->count;
    $total_chars += $element->chars;
}

// Final summary
print_box("Total elements", $total_elements);
print_box("Total characters", $total_chars);
```

Listing 7.3: XMLStats – Erfassung von statistischen Daten über ein XML-Dokument

Diese Anwendung verwendet Expat, um statistische Daten über ein XML-Dokument zu erfassen. Für jedes Element gibt dieser eine Reihe von Informationen aus:

- Häufigkeit dieser Information im Dokument
- Anzahl der Zeichendaten in diesem Element
- Alle gefundenen Elternelemente
- Alle Tochterelemente

Hierfür muss das Skript zumindest das Elternelement des aktuellen Elements kennen. Wenn Sie einen normalen XML-Parser verwenden, ist dies nicht möglich. Sie erhalten nur Ereignisse für das aktuelle Element. Es werden aber keine Kontextinformationen aufgezeichnet. Deshalb müssen wir eine eigene Stapelstruktur errichten. Wir hätten einen FIFO-Stapel (First In, First Out) mit zwei Elementen verwenden können; aber um Ihnen einen besseren Eindruck über die Elementverschachtelung in einer Datenstruktur zu vermitteln, haben wir uns für einen FILO-Stapel (First In, Last Out) entschieden. Dieser Stapel, der ein normales Array ist, enthält alle derzeit offenen Elemente. In der Bearbeitungsroutine für die offenen Elemente wird das aktuelle Element mit `array_push()` oben auf den Stapel gelegt. Analog dazu entfernt die Funktion der Endelementroutine das oberste Element mit `array_pop()`.

Hinweis: `array_pop()` und `array_push()` sowie viele weitere nützliche Funktionen, die sich mit Arrays befassen, wurden erst in PHP 4.0 hinzugefügt. Wir wollten sie nach PHP 3.0 portieren, aber es ist schwierig, sie im eigentlichen PHP zu integrieren (um sie wieder nach PHP 3.0 zurück zu portieren). Dies liegt an der Art und Weise wie `unset()` arbeitet. Um ein Element vom Stapel zu entfernen, brauchen Sie einen Befehl wie den folgenden:

```
unset($array[count($array) - 1]);
```

Wenn dies funktionierte, wäre es einfach, `array_pop()` zu implementieren. – es funktioniert jedoch nicht. Bei PHP hinterlässt `unset()` Löcher im Array. Es setzt den »Indexzähler« nicht zurück. Dies können Sie leicht selbst überprüfen:

```
$array = array("a");
unset($array[0]);
$array[] = "a"; _var_dump($array);
```

Das Element `a` hat jetzt nicht den erwarteten Schlüssel `0`, sondern `1`. Dies führt zu fragmentierten Arrays – was für einen Stapel sehr ungünstig ist. Dieses Verhalten liegt in den anderen Elementen in dem Array begründet. Wenn das Loch entfernt würde, müsste das Array neu organisiert werden, was in vielen Situationen unerwünscht wäre. Um dieses Problem zu umgehen, bräuchten wir eine `array_compact()`-Version, die es zum Zeitpunkt der Bucherstellung in PHP nicht gab. Daraus können wir nur folgenden Schluss ziehen: Verwenden Sie PHP 4.0. In der PHP-Version 3.0 unseres Beispiels (siehe CD-ROM) mussten wir die Variable `$depth` verwenden, um die Elementverschachtelung

manuell zu überwachen. Dies machte eine weiter globale Variable erforderlich und ist nicht so elegant wie as `array_pop()` und `array_push()`, aber es funktioniert.

Um Informationen über die einzelnen Elemente zu erfassen, müssen sich die Skripte alle Stellen merken, an denen die Elemente auftauchen. Wir verwenden die globale Array-Variable `$elements`, um die verschiedenen Elemente des Dokuments zu erfassen. Die Arrayeinträge sind Instanzen der Klasse `element`, die vier Eigenschaften hat (Klassenvariablen).

Eigenschaft	Beschreibung
`count`	Häufigkeit, mit welcher das Element im Dokument gefunden wurde
`chars`	Anzahl der Bytes in Form von Zeichendaten in diesem Element
`parents`	Elternelemente
`childs`	Tochterelemente

Wie Sie sehen, ist es kein Problem, Klasseninstanzen in einem Array zu speichern.

> **Tipp:** Eine spezifische Sprachfunktion von PHP ist, dass Sie Klassenstrukturen mit der Schleife `while(list() = each())`, die wir in Kapitel 1 »Entwicklungskonzepte« vorgestellt haben, genau so durchsuchen können, wie assoziative Arrays. Sie gibt alle Klassenvariablen und die Namen von Methoden als Zeichenketten aus.

Jedes Mal, wenn ein Element gefunden wurde, wird das Element `count` in dem entsprechenden `elements` im Array-Eintrag hochgesetzt. Im Eintrag des Elternelements (Eltern bezieht sich auf das zuletzt geöffnete Elementtag) wird der Name des aktuellen Elements dem Array `childs` angehängt. Das Elternelement wird mit dem Schlüssel `parents` zum Array-Eintrag hinzugefügt. Der übrige Programmcode durchläuft das Array `elements` und seine Unterarrays mehrfach, um statistische Werte anzuzeigen. Dies ergibt zwar eine schöne Ausgabe, aber der Code an sich ist weder besonders elegant, noch verfügt er über clevere Tricks. Es ist eine einfache Schleife, die Sie möglicherweise jeden Tag verwenden, nur um die Aufgabe zu erledigen.

7.2.7 DOM – das Dokumentenobjektmodell

Die andere Hauptfamilie der XML-Parser sind jene, die einen Zugriff auf eine Document-Object-Model(DOM)-Struktur ermöglichen. Wie Sie gesehen haben, müssen Sie bei ereignisbasierten Parsern häufig Ihren eigenen Datenstrukturen errichten. Der DOM-Ansatz vermeidet dies, indem er im Haupt-

speicher eine eigene Struktur generiert. Anstatt auf bestimmte Ereignisse zu reagieren, arbeiten Sie mit dieser Struktur, um das Dokument zu verarbeiten. Während ereignisbasierte Parser ein XML-Dokument in kleinen Teilen lesen und damit den Speicherplatz für die Analyse reduzieren, was gleichzeitig die Leistungsfähigkeit steigert, müssen DOM-Parser eine Darstellung des gesamten Dokuments im Speicher erzeugen. Dafür wird mehr Speicherkapazität benötigt. Bedenken Sie dies, wenn Sie mit großen Dokumenten arbeiten.

Die DOM-Ebene 1.0 wurde im Oktober 1998 von der (inzwischen wahrscheinlich bekannten) W3C-Organisation als Standard (W3C-Empfehlung) definiert. Möglicherweise haben Sie bereits in einem anderen Kontext vom DOM-Standard gehört. Dieser Begriff wird üblicherweise auch verwendet, um das Objektmodell von HTML-Seiten zu beschreiben, auf die mit JavaScript zugegriffen werden kann. Um beispielsweise den Wert eines Formularfeldes zu lesen, könnten Sie folgende JavaScript-Programmzeile verwenden:

```
fieldvalue = document.myform.myfield.value;
```

Beachten Sie die Hierarchie in dieser Anweisung. `document` ist das Stammelement, während `myform` ein HTML-Formular kennzeichnet, in welchem `myfield` ein Textfeld ist. Genau genommen ist das HTML-DOM eine Erweiterung des Kern-Dokumenten-Objekt-Modells, das vom W3C definiert wurde. Der DOM-Kern enthält die Funktionen, die für XML-Dokumente verwendet werden. Außerdem dient er als Grundlage für das HTML-DOM. Er besteht aus einer Sammlung von Objekten, die Sie für den Zugriff und die Bearbeitung von Daten und Auszeichnungen in einem XML-Dokument benutzen können. Er dient zur Definition folgender Punkte:

- einer Reihe von Objekten zur Darstellung der kompletten Struktur eines XML-Dokuments
- eines Modells für die Kombinationsmöglichkeiten dieser Objekte
- einer Schnittstelle für den Zugriff und die Überarbeitung dieser Objekte

Durch Abstrahierung des Dokuments erzeugt das DOM eine Baumstruktur, welche die Beziehung von Eltern- und Tochterknoten aufzeigt, sowie Methoden, wie `getAttribute()` für die Knoten. Kurzum, DOM stellt eine standardobjektorientierte und verzeichnisstruktur-ähnliche Schnittstelle für XML-Dokumente zur Verfügung.

Die DOM-Spezifikation ist unabhängig von einer Programmiersprache. Sie empfiehlt die Entwicklung einer objektorientierten Anwendung, wozu eine Sprache mit zumindest einfachen objektorientierten Funktionen erforderlich ist. Sie definiert eine Reihe von Knotentypen (Schnittstellen), die zusammengenommen das vollständige Dokument bilden. Einige Knotentypen haben möglicherweise Tochterknoten, andere Blattknoten, unter denen keine weiteren Knoten folgen. Auf diese Knotentypen werden wir im Folgenden ausführlich eingehen, da sie in der ursprünglichen W3C-Spezifikation hervorgeho-

ben wurden. Eine ausführliche Beschreibung aller Methoden und Attribute der einzelnen Instanzen finden Sie in der Spezifikation.

Document

Die Schnittstelle `Document` ist der Stammknoten des Strukturbaums. Diese Schnittstelle kann nur ein Element enthalten, und zwar das Stammelement des XML-Dokuments. Darüber hinaus kann es die mit diesem Dokument verknüpfte Dokumententypdeklaration enthalten (organisiert in der Schnittstelle `DocumentType`) und, sofern verfügbar, Verarbeitungsbefehle und Kommentare außerhalb des Stammelements.

Da sich alle anderen Knoten unterhalb des Knotens `Document` befinden, besitzt die Schnittstelle `Document` eine Reihe von Verfahren, um Unterknoten zu erzeugen. Mithilfe dieser Funktionen können Sie ein vollständiges XML-Dokument mittels eines Programms erstellen. In der Spezifikation ist auch eine Methode namens `getElementsByTagName()` definiert. Diese dient dazu, alle Elemente mit einem bestimmten Tag-Namen im Dokument auszulesen.

DocumentFragment

Der Knoten `DocumentFragment` stellt einen Teil eines komplexen XML-Dokuments dar. Häufig ist es notwendig, Teile eines Dokuments neu anzuordnen oder einen Teil zu extrahieren. Dafür benötigen Sie ein leichtes Objekt, um das entnommene Fragment zu speichern. Stellen Sie sich vor, Sie wollten eine einzelne Buchdatei aus vielen verschiedenen Kapiteldateien erstellen. Jedes Kapitel könnte in einem `DocumentFragment`-Objekt abgelegt sein und in die Dokumentenstruktur des Buches eingefügt werden Wenn Sie keine Möglichkeit haben, Fragmente eines Dokuments zu organisieren, müssten Sie alle Elemente aller Kapitel nacheinander in dem Buchdokument hinzufügen.

Die Spezifikation macht es sogar noch einfacher, indem sie festlegt, dass beim Einfügen von `DocumentFragment` in einen Knoten nicht `DocumentFragment` selbst, sondern nur die Tochterknoten von `DocumentFragment` in den Knoten eingefügt werden.

DocumentType

Der Knoten `DocumentType` enthält, sofern vorhanden, die Dokumententypdeklaration eines Dokuments. Diese Schnittstelle hat Nur-Lese-Status und kann bisher nicht vom DOM geändert werden.

Element

Jedes Element in einem Dokument wird durch einen `Element`-Knoten repräsentiert. Den Namen eines Elements können Sie mit dem Parameter `tagName` ermitteln. Außerdem definiert diese Schnittstelle eine Reihe von Funktionen, um Elementattribute zu setzen und zu erfassen und um auf Unterelemente zuzugreifen.

Attr

Der Knoten Attr repräsentiert ein Elementattribut in einem Element-Objekt. Den Namen und den Wert des Attributs ermitteln Sie mit den Variablen name und value dieser Schnittstelle. Die Variable specified gibt an, ob der Wert dieses Attributs vom Benutzer angegeben wurde oder ein Standardwert ist, der in der DTD angegeben ist.

EntityReference

Dieser Knoten repräsentiert einen Entitätenverweis, der im XML-Dokument gefunden wurde. Bedenken Sie, dass Zeichenverweise, (z.B. <) vom XML-Parser expandiert werden und daher nicht als EntityReference-Knoten dargestellt werden.

Entity

Dieser Knoten repräsentiert eine analysierte oder nicht analysierte Entität.

ProcessingInstruction

Der Knoten ProcessingInstruction repräsentiert einen Verarbeitungsbefehl in einem Dokument. Er besitzt zwei Attribute, nämlich target (das Ziel des Verarbeitungsbefehls) und data (den Inhalt).

Comment

Die Schnittstelle CharacterData repräsentiert den Inhalt eines Kommentars, d.h. alle Zeichen zwischen <!-- und -->. Er verfügt über keine weiteren Attribute oder Methoden.

Text

Die Schnittstelle Text CharacterData repräsentiert die Zeichendaten (textlichen Inhalt) eines Element- oder Attr-Knotens. Die Schnittstelle Text hat keine Attribute, aber eine Methode namens splitText(). Sie teilt einen Text-Knoten in zwei Hälften, was sich beispielsweise bei der Neuanordnung des Inhalts als nützlich erweisen kann.

CDATASection

Die Schnittstelle CDATASection erbt die Schnittstelle Text (und damit die Schnittstelle CharacterData) und enthält den CDATA-Abschnitt.

Notation

Dieser Knoten stellt eine in der Dokumententypdeklaration deklarierte Notation dar.

Basisschnittstellen

Alle oben genannten Objekte erben die Schnittstelle `Node`, die den primären Basistyp des DOM darstellt. Er stellt einen einzelnen Knoten in der Dokumentenverzeichnisstruktur dar. Die Schnittstelle `Node` definiert die Attribute und Methoden, die Sie am häufigsten verwenden, wenn Sie mit dem DOM arbeiten. Um beispielsweise ein Dokument zu durchsuchen, würden Sie das Attribut `childNodes` verwenden, das alle Tochterknoten enthält, sowie das Attribut `nextSibling`, das den nächsten Knoten auf der gleichen Ebene enthält. Um die Verzeichnisstruktur zu ändern, können Sie Methoden, wie `appendChild()` und `removeChild()` verwenden.

Die einzigen Objekte, die nicht direkt von der Schnittstelle `Node` abgeleitet wurden, sind `CDATASection`, `Text` und `Comment`. `Text` und `Comment` werden von der Schnittstelle `CharacterData` abgeleitet und `CDATASection` erbt `Text`. Die Schnittstelle `CharacterData` erweitert `Node` um eine Reihe von Attributen und Methoden für den Zugriff auf Zeichendaten. Mit `substringData()` können Sie beispielsweise einen Teil der Zeichendaten extrahieren.

Beispiel: Analyse eines kurzen Dokuments mit dem DOM

Den besten Eindruck über die konkrete Funktionsweise des DOMs erhalten Sie, indem wir uns ansehen, wie ein einfaches XML-Dokument von einem DOM-kompatiblen Prozessor verarbeitet wird. Dazu erstellen wir ein kurzes Buchdokument:

```
<?xml version="1.0"?>
<!DOCTYPE book SYSTEM "docbookx.dtd">
<book>
    <title>
        Cutting-Edge Applications
    </title>
    <para language="en">
        Sample paragraph.
    </para>
</book>
```

Eine DOM-Repräsentation dieses Dokument wird in einer hierarchischen Struktur wie jener in Abbildung 7.5 organisiert. In einer DOM-kompatiblen API sollte der Code so ähnlich aussehen wir der folgende Pseudocode:

```
/ Construct Document class instance
$doc = new Document("file.xml");

// Output the root element's name
printf("Root element: %s<p>", $doc->documentElement->tagName);

// Get all elements below the root node
```

```
$node_list = $doc->getElementsbyTagName("*");

// Traverse the returned node list
for($i=0; $i<$node_list->length; $i++)
{
    // Create node
    $node = $node_list->item($i);

    // Output node name and value
    printf("Node name: %s<br>", $node->nodeName);
    printf("Node value: %s<br>", $node->nodeValue);
}
```

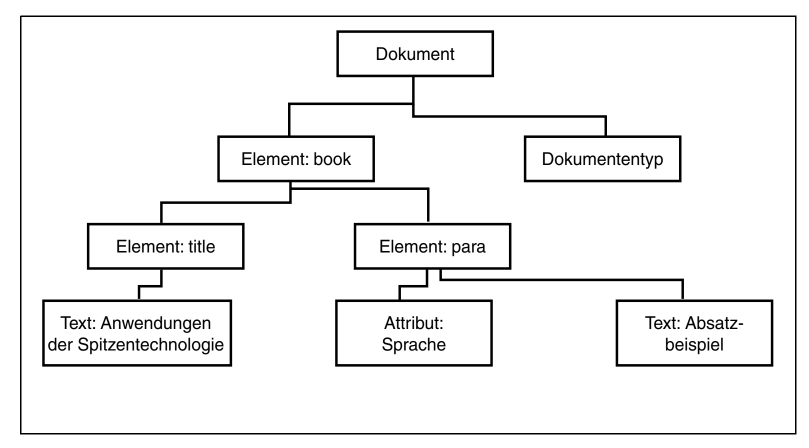

Abbildung 7.5: DOM-Struktur

7.2.8 LibXML – ein DOM-basierter XML-Parser

Seit der Version 4.0 ist in PHP ein neuer XML-Parser intergriert: LibXML. Dieser Parser wurde ursprünglich von Daniel Veillard für das Gnome-Projekt entwickelt, um einen DOM-fähigen Parser für einen komplexeren Datenaustausch zu erhalten. Uwe Steinmann integrierte ihn schließlich in PHP.

Auch wenn die interne Darstellung eines Dokuments in LibXML jener der DOM-Schnittstelle ähnelt, wäre es irreführend, LibXML als einen DOM-Parser zu bezeichnen. Die Analyse und die Verwendung des DOMs geschehen zu zwei ganz verschiedenen Zeitpunkten. Sie könnten genauso gut eine API oberhalb von Expat generieren, um eine DOM-Schnittstelle bereitzustellen. Zugegeben, die LibXML-Bibliothek erleichtert diese Aufgabe. Sie müssen lediglich die API ändern, um sie der DOM-Spezifikation anzupassen. In Gnome gibt es sogar ein Gdome-Modul, das eine DOM-Schnittstelle für LibXML bereitstellt.

> **Hinweis:** Als dieses Buch erstellt wurde, war die LibXML-API noch nicht hundertprozentig in PHP integriert. Sie war noch instabil und enthielt Bugs. Trotzdem konnte man bereits die immensen Vorteile sehen, welche die fertige LibXML-API bieten wird. Deswegen haben wir beschlossen, hier die grundlegenden Prinzipien zu erläutern und einige Beispiele vorzustellen. Wenn Änderungen auftreten, werden wir diese auf der Website des Buches dokumentieren.

Überblick

Die meisten Entwickler werden uns zustimmen, dass ein XML-Dokument am besten durch eine Baumstruktur dargestellt wird. LibXML bietet eine schöne API, um aus einer XML-Datei Bäume und DOM-ähnliche Datenstrukturen zu generieren. Wenn Sie ein Dokument mit LibXML analysieren, erstellt PHP eine Reihe von Klassen, mit denen Sie direkt arbeiten können. Indem Sie die Funktionen dieser Klassen aufrufen, können Sie auf alle Ebenen der Struktur zugreifen und das Dokument verändern.

Die beiden wichtigsten Objekte, die Ihnen bei der Arbeit mit LibXML begegnen werden, sind Dokumente und Knotenobjekte.

XML-Dokumente

Das abstrakte XML-Dokument wird in einem Dokumentenobjekt dargestellt. Solch ein Objekt wird über die Funktionen `xmldoc()`, `xmldocfile()` und `new_xmldoc()` erzeugt.

Die Funktion `xmldoc()` besitzt nur ein Argument, welches das XML-Dokument als Zeichenkette angibt. Die Funktion `xmldocfile()` verhält sich ähnlich, gibt aber im Argument einen Dateinamen an. Um ein neues leeres XML-Dokument zu erzeugen, können Sie die Funktion `new_xmldoc()` verwenden.

Alle drei Funktionen geben ein Dokumentenobjekt zurück, dem vier Methoden und eine Klassenvariable zugeordnet sind:

- `root()`
- `add_root()`
- `dtd()`
- `dumpmem()`
- `version`

Die Funktion `root()` gibt ein Knotenobjekt zurück, welches das Stammelement des Dokuments enthält. Bei leeren Dokumenten, die Sie mit `new_xmldoc()` erzeugt haben, können Sie mit `add_root()` ein Stammelement hinzufügen. Auch hier wird ein Knotenobjekt zurückgegeben. Beim Aufruf als Klassenmethode erwartet die Funktion `add_root()` als erstes Argument den Namen des

Elements. Sie können sie auch als globale Funktion aufrufen. In diesem Fall müssen Sie jedoch als ersten Argument die Instanz der Dokumentenklasse übergeben und als zweites Element den Namen des Stammelements.

Die Funktion `dtd()` gibt ein DTD-Objekt ohne Methoden, aber mit den Klassenvariablen `name`, `sysid` und `extid` zurück. Der Name der DTD ist stets identisch mit dem Namen des Stammelements. Die Variable `sysid` gibt den Systembezeichner an (z.B. `docbookx.dtd`), während die Variable `extid` den externen oder öffentlichen Bezeichner enthält. Zur Umwandlung der speicherinternen Struktur in eine Zeichenkette können Sie die Funktion `dumpmem()` verwenden. Die Klassenvariable `version` gibt die XML-Version des Dokuments an; in der Regel ist diese 1.0.

Mit diesen Erläuterungen können wir jetzt zu einem ersten einfachen Beispiel übergehen. Konstruieren wir einmal das XML-Dokument `Hello World` mit LibXML:

```
$doc = new_xmldoc("1.0");
$root = $doc->add_root("greeting");
$root->content = "Hello World!";
print(htmlspecialchars($doc->dumpmem()));
```

Damit erhalten wir folgendes wohlgeformtes XML-Dokument:

```
<?xml version="1.0"?>
<greeting>Hello World!</greeting>
```

Das Beispiel zeigt auch eine Eigenschaft, die Sie noch nicht kennen, nämlich den Zugriff auf den Inhalt eines Knotenobjekts.

Knoten

Das Tao Te King sagt, alles ist Tao. Bei der XML-Analyse ist alles ein Knoten. Elemente, Attribute, Text, Verarbeitungsbefehle usw. – aus Sicht eines Programmierers können Sie alle auf eine ähnliche Weise behandeln, da alle Knoten sind.

Wie wir bereits erwähnt haben, können Knoten die einfachste atomare Struktur eines XML-Dokuments wiedergeben. Einem Knotenobjekt sind folgende Funktionen und Variablen zugeordnet:

- parent()
- children()
- new_child()
- getattr()
- setattr()
- attributes()
- type
- name
- wenn verfügbar, content

Mit diesen Funktionen und Variablen können Sie alle verfügbaren Informationen über einen Knoten ermitteln. Sie können auf seine Attribute, seine Tochterknoten (sofern vorhanden) und den Elternknoten zugreifen. Darüber hinaus können Sie die Baumstruktur verändern, indem Sie Tochterknoten hinzufügen oder Attribute setzen. Listing 7.4 zeigt, wie diese Funktionen arbeiten. Hierbei handelt es sich um den XML-Pretty Printer, den wir bereits im Expat-Abschnitt erwähnt haben und der nach LibXML portiert wurde. Statt die Funktionen der Bearbeitungsroutine zu registrieren, wendet er je nach Knotentyp eine andere Formatierung an. Jedem Knoten ist ein Typ zugeordnet. Der Typbezeichner ist eine PHP-Konstante, und der Quellcode dieses Beispiels zeigt die vollständige Liste aller Typen. Mithilfe der Funktion children(), welche die Tochterelemente des Knotens (als Knotenobjekte) zurückgibt, können Sie das Dokument mehrfach durchlaufen. Im Beispiel wird eine rekursive Schleife durchlaufen, indem die Funktion output_node() immer wieder aufgerufen wird.

```
// Define tab width
define("INDENT", 4);

function output_node($node, $depth)
{
    // Different action per node type
    switch($node->type)
    {
        case XML_ELEMENT_NODE:
            for($i=0; $i<$depth; $i++) print(" ");

            // Print start element
            print("<span class='element'>&lt;");
            print($node->name);

            // Get attribute names and values
            $attribs = $node->attributes();
            if(is_array($attribs))
            {
```

```
            while(list($key, $value) = each($attribs))
            {
                print(" $key = <span class='attribute'>$value</span>");
            }
        }

        print("&gt;</span><br>");

        // Process children, if any
        $children = $node->children();
        for($i=0; $i < count($children); $i++)
        {
            output_node($children[$i], $depth+INDENT);
        }

        // Print end element
        for($i=0; $i<$depth; $i++) print(" ");
        print("<span class='element'>&lt;/");
        print($node->name);
        print("&gt;</span><br>");
        break;
    case XML_PI_NODE:
        for($i=0; $i<$depth; $i++) print(" ");
        printf("<span class='pi'>&lt;?%s %s?&gt;</span><br>",
                                        ➥$node->name, $node->content);
        break;
    case XML_COMMENT_NODE:
        for($i=0; $i<$depth; $i++) print(" ");
        print("<span class='element'>&lt;!-- </span>");
        print($node->content);
        print("<span class='element'> --&gt;</span><br>");
        break;
    case XML_TEXT_NODE:
    case XML_ENTITY_REF_NODE:
    case XML_ENTITY_REF_NODE:
    case XML_DOCUMENT_NODE:
    case XML_DOCUMENT_TYPE_NODE:
    case XML_DOCUMENT_FRAG_NODE:
    case XML_CDATA_SECTION_NODE:
    case XML_NOTATION_NODE:
    case XML_GLOBAL_NAMESPACE:
    case XML_LOCAL_NAMESPACE:
    default:
        for($i=0; $i<$depth; $i++) print(" ");
        printf("%s<br>", isset($node->content) ? $node->content : "");
  }
}
```

```
// Output stylesheet
?>
<style type="text/css">
<!--
.xml {  font-family: "Courier New", Courier, mono;
        font-size: 10pt; color: #000000}
.element {  color: #0033CC}
.attribute {  color: #000099}
.pi {  color: #990066}
-->
</style>
<span class="xml">
<?

// Process the file passed as first argument to the script
$file = "test.xml";

// Initial indenting
$depth = 0;

// Check if file exists
if(!file_exists($file))
{
    die("Can't find file \"$file\".");
}

// Create xmldoc object from file
$doc = xmldocfile($file) or die("XML error while parsing file \"$file\"");

// Access root node
$root = $doc->root();

// Start traversal
output_node($root, $depth);

// End stylesheet span
print("</span>");
```

Listing 7.4: XML-Pretty Printer – Beispiel für die Verwendung der LibXML-Funktionen

Einer der großen Vorteile, die LibXML im Vergleich zu Expat hat, besteht darin, dass Sie diesen Parser auch zum Konstruieren von XML-Dokumenten verwenden können. Damit ersparen Sie sich, eigene Routinen zur XML-Erzeugung schreiben zu müssen. Außerdem müssen Sie beispielsweise keine Verschachtelungsebenen mehr erfassen, um Tags korrekt zu schließen. Listing 7.5 setzt unser voriges Hello-World-Beispiel fort und generiert ein

komplettes RSS-Dokument (RSS steht für Rich-Site-Summary, ein XML-Formular zur Bereitstellung von Inhalten für Websites). Das Beispiel verwendet `setattr()`, um Attribute zu einem Element hinzuzufügen, und `new_child()`, um Elemente einem Knoten hinzuzufügen. Haben Sie bemerkt, wie `new_child()` verwendet wird? Die Funktion gibt ein Knotenobjekt zurück, aber es steht Ihnen frei, den ausgegebenen Wert einfach zu verwerfen, wenn Sie ihn nicht brauchen. Sie ordnen ihn einfach nur dann einer Variablen zu, wenn Sie Tochterelemente zu dem gerade erzeugten Knoten hinzufügen möchten.

```
$doc = new_xmldoc("1.0");

$root = $doc->add_root("rss");
$root->setattr("version", "0.91");

$channel = $root->new_child("channel", "");
$channel->new_child("title", "XML News and Features from XML.com");
$channel->new_child("description", "XML.com features a rich mix of
➥information and services for the XML community.");
$channel->new_child("language", "en-us");
$channel->new_child("link", "http://xml.com/pub");
$channel->new_child("copyright", "Copyright 1999, O'Reilly and Associates
➥and Seybold Publications");
$channel->new_child("managingEditor", "dale@xml.com (Dale Dougherty)");
$channel->new_child("webMaster", "peter@xml.com (Peter Wiggin)");

$image =$channel->new_child("image", "");
$image->new_child("title", "XML News and Features from XML.com");
$image->new_child("url", "http://xml.com/universal/images/xml_tiny.gif");
$image->new_child("link", "http://xml.com/pub");
$image->new_child("width", "88");
$image->new_child("height", "31");

print(htmlspecialchars($doc->dumpmem()));
```

Listing 7.5: Verwendung von LibXML-Routinen zum Erstellen von XML-Dokumenten

XML-Bäume

Mit den oben vorgestellten Methoden werden separate Objekte für das Dokument und die einzelnen Knoten erstellt. Dies eignet sich sehr gut für das Durchschleifen durch das Dokument, wie das Beispiel mit dem XML-Pretty-Printer zeigt. Aber der Zugriff auf einzelne Elemente wird dadurch leicht erschwert. Erinnern Sie sich noch an unser Beispieldokument »Hello World«?

```
<?xml version="1.0"?>
<greeting>Hello World!</greeting>
```

Um auf den Inhalt des Stammelements zuzugreifen, müssen Sie folgenden Code verwenden:

```
// Create xmldoc object from file
$doc = xmldocfile("test.xml") or die("XML error while parsing file \"$file\
➥"");

// Access root node
$root = $doc->root();

// Access root's children
$children = $root->children();

// Print first child's content
print($children[0]->content);
```

Dieses Beispiel hat nur eine Tiefe. Stellen Sie sich vor, wie Sie fortfahren müssten, wenn Sie stärker verschachtelte Elemente hätten. Wenn Sie das für einen zu großen Aufwand halten, können wir Ihnen nur zustimmen. Glücklicherweise war Uwe Steinmann derselben Meinung. Deshalb stellte er eine elegantere Methode für den willkürlichen Zugriff auf Dokumentenelemente zur Verfügung: `xmltree()`. Diese Funktion generiert eine Struktur der PHP-Objekt, die das gesamte XML-Dokument darstellen. Wenn Sie ihr im ersten Argument eine Zeichenkette übergeben, die ein XML-Dokument enthält, gibt die Funktion ein Dokumentenobjekt zurück. Das Objekt unterscheidet sich jedoch von dem weiter oben beschriebenen. Es erlaubt keinen Aufruf von Funktionen, aber setzt Eigenschaften genauso. Anstatt eine Liste der Tochterelemente mit dem Aufruf `children()` zusammenzustellen, sind die Tochterelemente bereits in der Struktur vorhanden (in der Klassenvariablen `children`). Dies erleichtert den Zugriff auf Elemente in jeder Tiefe. Auf den Inhalt des Elements `greeting` können Sie demnach mit dem folgenden Aufruf zugreifen:

```
// Create xmldoc object from file
$doc = xmldocfile(join("", file($file)) or die("XML error while parsing
➥file \"$file\"");

      print($doc->root->children[0]->content);
```

Das sieht sehr viel besser aus. Wenn Sie die Struktur, die mit `xmltree()` zurückgegeben wurde, mit `var_dump()` ausdrucken, erhalten Sie folgende Ausgabe:

```
object(Dom document)(2) {
  ["version"]=>
  string(3) "1.0"

  ["root"]=>
  object(Dom node)(3) {
    ["type"]=>
```

```
    int(1)

    ["name"]=>
    string(8) "greeting"

    ["children"]=>
    array(1) {
      [0]=>
      object(Dom node)(3) {
        ["name"]=>
        string(4) "text"

        ["content"]=>
        string(12) "Hello World!"

        ["type"]=>
        int(3)
      }
    }
  }
}
```

Wie Sie sehen, ist dies eine große Datenstruktur, die das gesamte Dokument darstellt. Die tatsächlichen Teile der Struktur sind immer noch Dokumente oder Objektknoten. Intern werden die gleichen Klassendefinitionen verwendet. Im Gegensatz zu Objekten, die mit `xmldoc()` und ähnlichen Funktionen erzeugt wurden, können Sie bei diesen Strukturen keine Funktionen aufrufen. Folglich hat die Struktur, die mit `xmltree()` zurückgegeben wird, zum jetzigen Zeitpunkt Nur-Lese-Status. Um XML-Dokumente zu konstruieren, brauchen Sie andere Methoden.

7.3 Datenaustausch mit WDDX

Nachdem Sie erfahren haben, wie Sie XML-Dokumente erzeugen und verarbeiten, ist es Zeit, Ihnen eine reale Anwendung zu präsentieren, die diese Technologie verwendet. Wie Sie wissen, ist XML plattform- und herstellerunabhängig und wird sowohl von PHP als auch anderen Programmiersprachen unterstützt. Warum sollten wir XML dann nicht einsetzen, um zwischen verschiedenen Plattformen oder verschiedenen Programmiersprachen zu kommunizieren? WDDX tut genau dies.

7.3.1 Datenaustausch über das Web (WDDX)

WDDX (Web Distributed Data eXchange) ist eine offene Technologie, die von Allaire Corporation entwickelt wurde. Es handelt sich hier um ein XML-Vokabular, das einfache und komplexe Datenstrukturen, wie Zeichenketten, Arrays und Datensätze, auf generische Art beschreibt, so dass diese zwischen verschiedenen Webskriptplattformen durch die Verwendung von HTTP hin- und hertransferiert werden können. PHP unterstützt WDDX, genau wie die meisten anderen bekannten Webskriptsprachen, wie z: B. Perl, ASP und Cold-Fusion.

7.3.2 Die Herausforderung

Nachdem Webanwendungen eine zunehmend größere Rolle in der Softwarewelt spielen, ist eine der größten Herausforderungen der Datenaustausch zwischen verschiedenen Programmierumgebungen. Das neue Web fördert verteilte und vernetzte Anwendungen. Es begann mit einfachen Websites, die statische Informationen enthielten. Sie wurden zu anspruchsvolleren, dynamischen Webanwendungen, und inzwischen beginnen diese Webanwendungen zusammenzuarbeiten, Daten auszutauschen und zusätzliche programmierbare Dienste anzubieten. Erst vor kurzem tauchten offene Standards wie XML auf. Sie verbreiteten sich so schnell, dass Hersteller gezwungen waren, diese Standards in ihrem Produkten zu übernehmen.

Durch die Erweiterung von Webanwendungen um offene Programmierschnittstellen für den Daenaustausch und die Modifikation von Daten wurde das Web erheblich verbessert. Plötzlich können Anwendungen miteinander reden – sie haben sich von geschlossenen proprietären Websites zu einer neuen Generation von verdrahteten Geschäftsanwendungen entwickelt.

Webanwendungen können eine API zur Verfügung stellen oder Datenstrukturen verfügbar machen. Damit wird der Datenaustausch zwischen verteilten Servern möglich. WDDX bietet eine mögliche Lösung zum Austausch von strukturierten Daten. WDDX besteht aus zwei Teilen: ein Teil befasst sich mit der Abstrahierung von Daten in eine XML-Darstellung, den sogenannten *WDDX-Paketen*; der andere Teil übersetzt Datenstrukturen in und aus einer Skriptumgebung sowie WDDX-Pakete.

WDDX dient nicht zum Fernaufruf von Prozeduren. Es gibt im reinen WDDX keine Möglichkeit, auf dem Server eine Funktion aufzurufen, die ein WDDX-Paket zurückgibt. Wie Sie Pakete übertragen, bleibt Ihnen überlassen. WDDX verfügt daher auch über keine Sicherheitsfunktionen. Es ist Ihre Aufgabe, einen Kommunikationskanal auszuwählen, der über die entsprechende Sicherheit verfügt und nur berechtigten Personen Zugriff auf den WDDX-Teil Ihres Systems gewährt.

7.3.3 Mögliche Szenarien

Ob zur Synchronisierung von Daten, zur Zentralisierung von Datendiensten oder zur Durchführung von Geschäftskommunikationen – der strukturierte Datenaustausch eröffnet eine neue Dimension der verteilten Datenverarbeitung. Stellen Sie sich z.B. vor, dass Sie auf einer E-Commerce-Website arbeiten. Wäre es nicht großartig, wenn Sie Kunden während des Versands eine Paketprotokollfunktion zur Verfügung stellen könnten? Dies ließe sich einfach realisieren, wenn UPS oder FedEx eine Möglichkeit böten, eine Abfrage nach einem versandten Artikel zu starten. Durch Einführung von formulargestützten und kundenorientierten Diensten (die meisten Kuriere ermöglichen ihren Benutzern, den Weg von Paketen zu verfolgen) für die verteilte Datenverarbeitung (Paketprotokollierung als programmierbare API) wurde eine neue Form der Anwendungsinteraktion ermöglicht.

In einem anderen Szenario könnte WDDX für die Interaktion von Server zu Server eingesetzt werden. Einige Dienste arbeiten möglicherweise am besten auf einer Windows NT Plattform, z.B. die Abfrage eines SAP-Backend über einen COM-Anschluss. Sie könnten ASP für diese Aufgabe verwenden und die Abfrageresultate als WDDX-Pakete an Ihre PHP-Anwendung auf dem UNIX-Webserver übergeben. Dort würden die Pakete transparent in PHP-Datenstrukturen, wie etwas Arrays, umgewandelt, ohne dass eine weitere Bearbeitung nötig ist.

7.3.4 Abstrahierung von Daten mit WDDX

Das Wichtigste zuerst: WDDX ist XML. Die möglichen WDDX-Strukturen werden in einer XML-DTD definiert, so dass gültige WDDX-Pakete stets wohlgeformte und möglicherweise gültige XML-Dokumente sind. Laut Spezifikation müssen die WDDX-Pakete nicht gültig sein, da sie keine Dokumententypdeklaration im Dokumentenprolog besitzen müssen.

Der grundlegende Gedanke ist, dass WDDX die Umwandlung einer beliebigen Standarddatenstruktur – Integer, Strings, Arrays usw. – in eine XML-Darstellung ermöglicht, die der WDDX-DTD entspricht. Diese Daten können dann über einen Kommunikationskanal übertragen werden, der XML-fähig ist: HTTP, FTP, Email usw. Die Daten können zu jedem Zeitpunkt aus dem WDDX-Format wieder in dieselbe Struktur wie zuvor zurück übertragen werden. Während des gesamten Konvertierungsprozesses, bleiben die Datentypen erhalten. Wenn Sie ein Array als WDDX-Paket serialisieren, wird es nach der Deserialisierung wieder ein Array sein.

Wenn dies alles wäre, wäre es nicht sehr aufregend. Immerhin können Sie eine beliebige Datenstruktur in PHP mithilfe von `serialize()` in eine proprietäre Form umwandeln und sie mit `deserialize()` wieder in ihr ursprüngliches Format zurückwandeln, und das so häufig Sie wollen. Der interessante Aspekt von WDDX ist, dass die serialisierte Darstellung ein XML-Format ist und die

Serialisierungs- und Deserialisierungsmodule klar von dieser getrennt sind. Alle Programmiersprachen, die WDDX unterstützen, können WDDX-Pakete serialisieren und deserialisieren. Serialisieren Sie ein PHP-Array in ein WDDX-Paket, deserialisieren Sie das Paket in ASP, und Sie haben wieder ein Array! Wie die Serialisierungs- und Deserialisierungsfunktion tatsächlich realisiert wird, ist unerheblich. In PHP haben Sie die `wddx_*()`-Funktionen, in ColdFusion können Sie das Tag `CFWDDX` verwenden, und in ASP steht ein COM-Element zur Verfügung.

Zugegeben, dies könnten Sie auch ohne WDDX bewerkstelligen. Es wäre jedoch recht kompliziert, insbesondere wenn Sie die Datentypen in den verschiedenen Programmierumgebungen erhalten möchten. Der Vorteil von WDDX liegt darin, dass es Sie von diesen Mühen befreit und eine flexible, offene und pragmatische Alternative bietet, um strukturierte Anwendungsdaten auszutauschen.

7.3.5 WDDX-Datentypen

Die meisten Programmiersprachen verwenden die gleiche Art von Datentypen, z.B. Strings oder Arrays. Der Zugriff auf diese Strukturen kann von Sprache zu Sprache variieren, aber der allgemeine Ansatz bleibt der gleiche. Ein String ist eine Folge von Zeichen. Ob Sie diese nun in einfache Anführungsstriche, doppelte Anführungsstriche oder gar nicht einschließen, macht vom Konzept her keinen Unterschied.

WDDX unterstützt diese generischen Datentypen. In der Version 1.0 der Spezifikation werden folgende Typen definiert:

- Null-Werte: Element `null`
- Boolesche Werte: `bool`
- Zahlen (ohne Unterscheidung zwischen ganzen Zahlen und Fließkommazahlen): `number`
- Strings: `string`
- Datum-/Zeitangabe: `dateTime`
- Indizierte Arrays: `array`
- Assoziative Arrays: `struct`
- Sammlungen von assoziativen Arrays, eine Gruppe benannter Felder mit der gleichen Anzahl an Datenzeilen: `recordset`
- Binäre Objekte, die z. Z. mit Base64 kodiert sind: `binary`

Alle WDDX-Pakete haben dasselbe XML-Format. Das Stammelement lautet `wddxPacket`. Dies hat ein optionales Versionsattribut, das die Version der WDDX-DTD angibt. Die Versionsnummer (derzeit `1.0`) ist gebunden an und

definiert in der DTD. Dies bedeutet, dass ein WDDX-Paket mit der Version 1.0 nur dann gültig ist, wenn es mit der WDDX-DTD 1.0 verwendet wird, und nicht mit 2.0 oder 0.9.

Direkt unter dem Element `wddxPacket` muss das Element `header` folgen. Der Kopf kann nur ein Kommentarelement enthalten und sonst nichts.

Das nächste erforderliche Element ist `data`, das den Hauptteil des Pakets kennzeichnet. Innerhalb dieses Elements muss genau ein Element des Typs `null`, `boolean`, `number`, `dateTime`, `string`, `array`, `struct`, `recordset` oder `binary` auftauchen. Listing 7.6 zeigt ein Beispiel für ein WDDX-Paket:

```
<wddxPacket version='1.0'>
    <header/>
    <data>
        <struct>
            <var name='string'>
                <string>This is a 'string'.</string>
            </var>
            <var name='int'>
                <number>42</number>
            </var>
            <var name='float'>
                <number>42.5</number>
            </var>
            <var name='bool'>
                <boolean value='true'/>
            </var>
            <var name='array'>
                <array length='3'>
                    <number>1</number>
                    <number>2</number>
                    <number>3</number>
                </array>
            </var>
            <var name='hash'>
                <struct>
                    <var name='foo'>
                        <string>bar</string>
                    </var>
                    <var name='baz'>
                        <string>fubar</string>
                    </var>
                </struct>
            </var>
```

```
        </struct>
    </data>
</wddxPacket>
```

Listing 7.6: Beispiel für ein WDDX-Paket

7.3.6 PHP und WDDX

Da WDDX zur Darstellung der Datenstrukturen XML verwendet, müssen Sie PHP sowohl mit XML als auch mit WDDX kompilieren, um es zu verwenden. Danach stehen Ihnen folgende Funktionen zur Verfügung:

- wddx_serialize_value()
- wddx_serialize_vars()
- wddx_packet_start()
- wddx_add_vars()
- wddx_packet_end()
- wddx_deserialize()

Mithilfe dieser Funktionen können Sie PHP-Variablen in WDDX-Pakete serialisieren und die WDDX-Pakete deserialisieren.

7.3.7 Die WDDX-Funktionen

Die WDDX-Funktionen bieten Ihnen drei Möglichkeiten, ein Paket zu konstruieren. Die einfachste Methode ist die Verwendung von wddx_serialize_value(). Diese Funktion besitzt eine Variable und einen optionalen Kommentar, und erzeugt hieraus ein WDDX-Paket:

```
$var = "This is a string.";
print(wddx_serialize_value($var, "This is a comment."));
```

Das ist schon alles. Dieser Programmabschnitt gibt das folgende WDDX-Paket aus.

```
<wddxPacket version='1.0'>
    <header>
        <comment>This is a comment.</comment>
    </header>
    <data>
        <string>This is a string.</string>
    </data>
</wddxPacket>
```

Hinweis: Zur besseren Lesbarkeit wurde dieses Paket manuell bearbeitet. Das von PHP erzeugte Paket enthält keine Leerräume und Einzüge.

Die beiden anderen Methoden serialisieren mehrere PHP-Variablen in ein WDDX-Element des Typs `struct`, ähnlich jenem, das in Listing 7.6 gezeigt wird. Die Funktion `wddx_serialize_vars()` besitzt eine beliebige Anzahl von String-Argumenten, welche die Namen der PHP-Variablen enthalten. Als Ausgabe gibt die Funktion die WDDX-Pakete als Strings zurück. Der Vorteil dieser Vorgehensweise besteht darin, dass Sie mehrere PHP-Variablen in einem WDDX-Paket serialisieren können. Beachten Sie jedoch, dass bei der Deserialisierung ein assoziatives Array erzeugt wird. Natürlich wurde die ursprüngliche PHP-Variable dazu vorher in einen WDDX-Tag `struct` umgewandelt. Im Programmcode könnte ein einfaches Beispiel für `wddx_serialize_vars()` folgendermaßen aussehen:

```
$string = "This is a string.";
$int = 42;
print(wddx_serialize_vars("string", "int"));
```

Die drei Funktionen `wddx_packet_start()`, `wddx_add_vars()` und `wddx_packet_end()` arbeiten grundsätzlich auf dieselbe Weise: Auch hier werden mehrere PHP-Variablen in ein WDDX-Tag des Typs `struct` umgewandelt. Der Unterschied ist, dass die Transaktion in drei Schritten vonstatten geht. Dies hat den Vorteil ,dass Sie PHP-Variablen zu einem WDDX-Paket langfristig hinzufügen können, z.B. während einer komplexen Berechnung. Im Gegensatz dazu arbeitet die Funktion `wddx_serialize_vars()` auf atomare Weise. Sie beginnt mit der Zusammenstellung eines Pakets, indem sie die Funktion `wddx_packet_start()` aufruft, die ein optionales Argument hat, den Header-Kommentar für das WDDX-Paket. Diese Funktion gibt einen Paketbezeichner zurück, ähnlich den Dateibezeichnern, die mit `fopen()` zurückgegeben werden. Dieser Bezeichner wird als erstes Argument der Funktion `wddx_add_vars()` verwendet. Die übrigen Argumente sind identisch mit jenen aus `wddx_serialize_vars()`: eine beliebige Anzahl von Strings, welche die Namen der PHP-Variablen angeben. Ein einfaches Beispiel:

```
$i1 = $i2 = $i3 = "Foo";
$packet_id = wddx_packet_start("This is a comment");
for($i=1; $i<=3; $i++)
{
    wddx_add_vars($packet_id, "i$i");
}
print(wddx_packet_end($packet_id));
```

Dieses Beispiel fügt in einer `for()`-Schleife dem WDDX-Paket einfach drei String-Variablen hinzu und erzeugt die folgende Ausgabe (auch hier wurden zur besseren Lesbarkeit entsprechende Einzüge hinzugefügt):

```
<wddxPacket version='1.0'>
    <header>
        <comment>This is a comment</comment>
    <data>
        <struct>
            <var name='i1'>
                <string>Foo</string>
            </var>
            <var name='i2'>
                <string>Foo</string>
            </var>
            <var name='i3'>
                <string>Foo</string>
            </var>
        </struct>
    </data>
</wddxPacket>
```

7.4 Zusammenfassung

Dieses Kapitel hat Ihnen alles Notwendige über die führenden Webanwendungen vermittelt. Wir haben Ihnen gezeigt, wie Sie Wissensdatenbanken konzipieren, generieren und einrichten, Daten darin richtig speichern und sie effizient auslesen. Wir haben übliche Datenformate und offene Standards für den Datenaustausch, Prozedurfernaufrufe und plattformunabhängige Datenspeicherung vorgestellt. Nachdem Sie die hier gezeigten Verfahren gelesen und verstanden haben, sind Sie in der Lage, die Schlüsselfunktionen der heutigen Spitzenanwendungen zu nutzen.

Pearson
Education

ADDISON-WESLEY

8 Fallstudien

Andere zu kennen, zeugt von Intelligenz;
Sich selbst zu kennen, von wirklicher Weisheit.

Die folgenden Abschnitte zeigen drei Fallstudien, in denen PHP auf großen Websites und in kommerziellen Produkten eingesetzt wird. Diese Fallstudien beschreiben Firmen, die sich nach sorgfältiger Auswertung der konkurrierenden Technologien für PHP entschieden haben und die in einem großen Teil ihrer täglichen Arbeit von PHP abhängen. Ein Großteil der Texte wurde von den Firmen selbst bereitgestellt. Diese Texte geben daher einen konkreten Einblick in die Entwicklungsprozesse und technischen Daten. Die Fallstudien belegen, dass PHP eine zuverlässige, hochentwickelte Lösung für Entwicklungen von serverseitigen Webanwendungen darstellt. Wir hoffen, dass sie Ihnen helfen, die Geschäftsleitung und Kollegen von PHPs Vorteilen zu überzeugen.

8.1 BizChek.com

Vielleicht wussten Sie, dass HotMail, eine Microsoft-Firma, auf ihren Servern FreeBSD verwendet. Es wäre zwar schön, wenn wir eine Fallstudie hätten, die zeigt, dass Microsoft PHP für seine webbasierten Mails verwendet, aber wir haben etwas zumindest Gleichwertiges: BizChek.com (`www.bizchek.com`). Der führende Anbieter webbasierter Emails für Firmen verwendet PHP. Mit Kunden, wie Budweiser und Micro Warehouse, sowie über zwei Millionen registrierten Benutzern ist BizChek.com einer der wichtigsten Akteure im kommerziellen Bereich.

Nicht genug damit – Mark Musone, CTO bei Chek Inc., der Mutterfirma von BizChek.com, ist ein aktiver PHP-Entwickler. Er hat solch nützliche Module, wie IMAP und MCAL, geschrieben oder mitgeschrieben, die inzwischen zur Standarddistribution von PHP gehören.

8.1.1 Web Mail

Mit mehr als 70 Angestellten, von denen acht Programmierer sind, ist BizChek ein wichtiger Anbieter im Bereich der webbasierten E-Commerce / Intranet-Marktes. Zusammen mit dem Rest von Chek Network rangiert Chek.com unter den ersten zwanzig der 100 Besten des Internets (`www.100hot.com`).

BizChek belegt eine einzigartige Position im Bereich der Email-Geschäftslösungen. Obwohl die Hauptanwendung ein selbständiges, webbasiertes Emailprogramm ist, bietet BizChek auch eine Reihe von Produktivitätswerkzeugen, welche das einzelne Geschäft kohärenter machen und gleichzeitig die Effizienz und Produktivität des gesamten Unternehmens erhöhen. BizChek ist mehr als nur ein Email-Produkt. Es kann einer Firma helfen, eine elektronische Identität aufzubauen. Die Firma kann sich und ihren Angestellten repräsentieren, indem sie in ihrer Email-Adresse anstelle einer generischen Adresse (AOL, HotMail usw.) ihren Firmennamen verwendet. Sobald eine Firma die materiellen und immateriellen Vorteile versteht, die sich aus einer webbasierten E-Commerce-Lösung ergeben, ist die Umwandlung oder Initiierung eines Dienstes einfach. Firmenbesitzer und Geschäftsführer werden sich zunehmend der Vorteile bewusst, die ein ausgelagertes webbasiertes Werkzeug zur elektronisfchen Kommunikation bringen kann.

BizChek bietet eine Reihe von Funktionen, einschließlich Email, Dokumenten / Datei-Sharing und Revision, Taskmanager, Kalenderdienste, ein Schwarzes Brett der Firma u. v. m. Nach einem finanziellen Vergleich wird sich herausstellen, dass das Outsourcing die Kosten der Firma in fast allen Fällen drastisch senkt. Ohne Investitionen in Hard- / Software und wegefallenem Wartungsaufwand aus Sicht der Informationstechnologie werden die Einsparungen real messbar. Outsourcing kann sich auch in einem verringerten Risiko ausdrücken, durch Bereitstellung einer professionellen und zuverlässigen Lösung, bei der die Virusüberwachung und die Vermeidung von Werbemails zentral verwaltet und kontrolliert werden. Dies ist ein weiterer immaterieller Vorteil, den ein ausgelagertes webbasiertes Kommunikationsmittel bietet.

8.1.2 Entscheidung für PHP

Seit der Einführung von BizChek im Jahre 1998 verwenden BizChek-Entwickler PHP, und zwar PHP 2.0. Auch wenn BizChek ganz und gar auf PHP ausgerichtet ist, bedeutet dies nicht, dass zuvor keine anderen Lösungen in Betracht gezogen wurden. Die Experten verbrachten mehrere Monate damit, verschiedene Alternativen zu testen und zu bewerten: ASP, Mod_Perl und Cold Fusion. Sie befanden, dass PHP die beste Wahl für BizChek sei, und seit Anbeginn richtete sich ihr Hauptaugenmerk auf PHP. Als junges dot-com-Unternehmen mit begrenzten Mitteln, das nichts als seine Arbeitskraft anzubieten hatte, waren sie von der Open-Source-Gemeinschaft sehr angetan. Obwohl PHP, wie viele andere Open-Source-Produkte, von großen Firmen nicht kommerziell unterstützt wird, belegten Marktstudien erst vor kurzem einen enormen Anstieg der PHP-Verwendung auf Websites und in anderen Produkten. Laut einer Webserver-Umfrage von Netcraft wurde PHP bei über elf Millionen insgesamt analysierten Servern im Februar 2000 auf über 1,4 Millionen Domänen verwendet. [1] Die BizChek-Entwickler sind der Mei-

nung, dass sich angesichts dieser Ergebnisse PHP und andere Open-Source-Produkte an die Spitze der entsprechenden Bereiche setzen und sich selbst gegen kommerzielle Mitbewerber aus etablieren Firmen behaupten werden.

Bei ihrer Suche nach einer Skriptsprache testeten die BizChek-Entwickler viele brauchbare Alternativen, die möglicherweise ihren Anforderungen genügten. Die ersten zwei Kandidaten waren Perl und verschiedene CGI-Skriptsprachen. Danach wurde Microsofts treibende Kraft ASP unter die Lupe genommen. Schließlich waren zwei Geheimtipps dran: Cold Fusion und PHP. Da eine ihrer Hauptanforderungen die serverseitige Programmierung und die Anbindung an eine Datenbank war, schlossen die BizChek-Entwickler von vornherein alle clientseitigen Skriptsprachen aus. Sie mochten Perl zwar wegen seiner vielseitigen Einsetzbarkeit, aber sie hatten den Eindruck, dass diese Sprache niemals für den Einsatz im Web optimiert wurde und bei der Verwaltung von starken Datenverkehr über mehrere Websites versagen würde. CGI- und Perl-Skripte können schnell den gesamten Speicher und die CPU belegen, was zu Leistungseinbußen führt. Da ASP, Cold Fusion und PHP mit dem Webserver zusammenarbeiten, ohne zusätzliche Prozesse zu erzeugen, kamen sie für die BizChek-Entwickler in die engere Wahl. Cold Fusion wäre vielleicht eine akzeptable Wahl gewesen, aber es war zu jener Zeit noch ein sehr neues Produkt. Man wusste zuwenig darüber, und deshalb ließen es die Entwickler wieder fallen. BizChek schätzte, dass bei der Wahl von Apache als Webserver Mod_Perl, das den Perl-Interpreter direkt in den Webserver integriert, Ergebnisse erzielen würde, die jenen von PHP und ASP nahe kämen. Auch wenn Mod_Perl vielleicht eine großartige Alternative gewesen wäre, hatten die BizChek-Entwickler die Befürchtung, dass es großen Webanwendungen nicht gewachsen wäre. Für sie war es eine zu allgemein einsetzbare Sprache, während sie PHP für eine sehr einfache und trotzdem leistungsfähige Variante hielten, die speziell für das Web entwickelt wurde. Man musste nicht durch irgendwelche Reifen springen, um PHP dazu zu bringen, »mit dem Web zusammenzuarbeiten«.

ASP bot eine Vielzahl von Komponenten anderer Hersteller, aber die Kosten für die Einführung und die Wartung waren besorgniserregend. Die Tatsache, dass ASP außerdem nur auf den Internet Information Server und Microsoft Windows-Plattformen beschränkt war, degradierten es in den Augen von BizChek noch weiter. Während sich ASP für die Datenverbindung hauptsächlich auf ODBC stützt, was sich als sehr sinnvoll erweisen kann, wenn Sie Ihre Datenbank jeden Monat wechseln, haben PHP und Perl eigene Datenbankverbindungen, was ihnen eine kleinen Vorteil verschafft. Die Fehlersuche, die, wie jeder Programmierer weiß, einen Großteil der Projektzeit in Anspruch nehmen und sich zu einem wahren Alptraum ausweiten kann, wenn die Präzision nicht hundertprozentig ist, war bei ASP ebenfalls schwieriger. PHP gibt präzise Fehlermeldungen aus, die eine effektive Hilfe für die Lösung des Problems darstellen.

Weitere Vorteile von PHP sah man in seiner klaren, C-ähnlichen Sprachsyntax, der Leichtigkeit, mit der es reinem HTML hinzugefügt werden kann, seiner Erweiterbarkeit, der Datenbankanbindung und dem Preis. Ins Gewicht fiel auch die Unterstützung, welche die PHP-Gemeinschaft bietet. Diese besteht aus einer umfangreichen Mailingliste, einem der besten Online-Dokumentationszentren, zahlreichen Online- und Hardcopy-Tutorials sowie einem regen Wissensaustausch zwischen Experten und Laien.

Ein weiterer Aspekt, den viele Leute bei Open-Source-Produkten häufig übersehen, ist, dass die Entwicklung permanent vorangetrieben wird, so dass in einigen Fällen monatlich neue Versionen freigegeben werden. Wenn Sie eine bestimmte Funktion brauchen, können Sie diese selbst hinzufügen, sofern Sie die Zeit und das notwendige Wissen besitzen. Dies bedeutet, dass Sie nicht mehr länger auf die neueste Version warten müssen, um festzustellen, ob diese Ihren Anforderungen gerecht wird. Einige BizChek-Programmierer gehören zum PHP-Entwicklungsteam und haben Programmcodes entwickelt, die aus ihren eigenen Bedürfnissen heraus entstanden. Die meisten Arbeiten zu PHP-Projekten werden zwar außerhalb des Betriebs durchgeführt, gelegentlich geschieht es aber auch während der Arbeitszeit. BizChek unterstützt diese Aktivität, da der entwickelte Programmcode direkt in ihr System zurückgeführt wird. Die BizChek-Entwickler haben zur Entwicklung von zahlreichen PHP-Modulen beigetragen, einschließlich MCAL, IMAP, FTP und Aspell. Ursprünglich wurden diese Module für die Unterstützung von BizChek entwickelt, da aber auch andere Programmierer von diesen Modulen profitieren konnten und die Entwickler das Bedürfnis hatte, der Gemeinschaft etwas zurückzugeben, wurden die Erweiterungen zum Kerncode von PHP hinzugefügt. Einige BizChek-Angestellte tragen auch zu kleineren Entwicklungen für Apache und Linux bei.

8.1.3 Begierig auf Updates

BizChek wurde bereits von der PHP-Version 2.0 auf die Version 3.0 umgestellt und kann auf PHP 4.0 umgestellt werden, sobald die Betatestphase abgeschlossen ist. Die Migration von 2.0 auf 3.0 sowie von 3.0 auf die 4.0-Betaversion ging ohne Komplikationen vonstatten und erforderte nur einige kleinere Änderungen.

Das gesamte System besteht aus ungefähr 200 Dateien und 25 bis 30 Tausend Programmzeilen – und wächst täglich. Da für jedes Update nur etwa 20 Programmzeilen geändert werden müssen, sind die BizChek-Entwickler mit der hervorragenden Rückwärtskompatibilität, die PHP bietet, sehr zufrieden.

Dem Vorhaben von BizChek, PHP 4.0 zu verwenden, das intern bereits getestet ist, steht nur die Besorgnis entgegen, Betaversionen auf einem Produktionsserver einzusetzen. Die Firma begrüßt die Updates im Bereich der Array-

Verarbeitung, Installation und die neuen Module der PHP 4.0-Betaversion. Das aktualisierte System zur Sitzungsunterstützung war eine der Funktionen, auf die BizChek gewartet hat, insbesondere da es bei der Einführung von BizChek noch kein PHPLib gab und sie nie auf diese Bibliothek umgestellt haben. Obwohl ihr eigenes Sitzungsverwaltungssystem zuverlässig arbeitet, werden sie auf das in PHP 4.0 integrierte Session Management umsteigen.

8.1.4 Schlußfolgerung

»PHP hat unsere Anforderungen genau erfüllt«, sagt CTO Mark Musone. »Es arbeitet mit Apache, MySQL und Oracle zusammen, um nur einige zu nennen. Und das auf sehr zuverlässige und leistungsfähige Weise, was zu effizienten dynamischen Webseiten mit sehr wenigen Problemen und nahezu keinen Ausfallzeiten führt. PHP ist für unser Geschäft ganz klar von Vorteil, und wir sind froh, dass wir es benutzen können.«

8.2 SixCMS

Professionelle Inhaltsverwaltungssysteme waren bisher die Domäne großer Firmen und sehr kostenintensiv, da sie mit Standardprodukten realisiert wurden, die in der Regel eine sechsstellige Summe kosteten. Wie würde Ihnen ein voll ausgestattetes Inhaltsverwaltungssystem (CMS) gefallen, das in PHP geschrieben ist? Six Offene Systeme GmbH (`www.six.de`) besitzt eines.

8.2.1 Firmenhintergrund

Das vor acht Jahren gegründete Unternehmen Six Offene Systeme GmbH (im Folgenden kurz als »Six« bezeichnet) entwickelte und vertrieb für verschiedene Industriezweige anspruchsvolle Datenbankanwendungen, die Open-Source-basierte Internet- und Intranet-Technologien verwendeten. Six hat sich auf die Entwicklung von individuellen Lösungen spezialisiert, die auf das Unternehmensziel zugeschnitten sind, wobei das Hauptargument auf dem Zielmarkt, der Produktpalette und den technischen Anforderungen des Kunden liegt. Die Bandbreite der Beratungsdienste und kundenspezifischen Lösungen von Six umfasst die Entwicklung von effektiven Geschäftsmodellen für die Online-Präsenz, die Definition von Meilensteinen mit einer differenzierten Strategie zur Implementierung, die Unterstützung für die Erarbeitung von mittel- und langfristigen Wirtschaftlichkeitsplänen, den Einsatz von Geschäftsprozessmodellierungen für die optimale Ressourcennutzung, die Formulierung von technischen Anforderungen sowie die Entwicklung von detaillierten Anforderungskatalogen. Das gesamte Wissen aus mehr als vier Jahren Programmiererfahrung hat sich nun in zwei standardisierten kommer-

ziellen Lösungen niedergeschlagen, die PHP einsetzen: SixCMS (ein Content Management-System) und SixAIM (ein System zur Kundenbetreuung und Fakturierung), die Six jetzt herausgebracht hat. Derzeit sind 27 Mitarbeiter bei Six beschäftigt, außerdem bietet die Firma eine hervorragende Umgebung und Möglichkeiten für Studenten aus verschiedenen Gebieten der Informationstechnologie und der neuen Medien, die ihr Praxissemester absolvieren wollen oder ihre Diplomarbeit schreiben.

Von den 27 Six-Mitarbeitern sind 20 direkt an der Entwicklung und Programmierung von Anwendungen beteiligt. Eine der Vorteile von Six besteht darin, dass ihre Entwickler ganz unterschiedliche Ausbildungs- und Erfahrungsniveaus mitbringen und aus diversen Gebieten wie Informatik, Physik, Medien, Kartographie, Agrikultur, Philosophie und Systemverwaltung kommen.

8.2.2 Open-Source-Geschäft

Six sieht beim Vergleich zu vielen der mehr proprietären Technologien immer noch einen klaren Vorteil in der Nutzung der Open Source/Systeme. Als Marktführer und Anbieter von Weltklasselösungen in Deutschland und Westeuropa hat Six im Januar 2000 eine Filiale in New York City eröffnet.

Von vielen Leuten wird die SixCMS-Anwendung als die Nummer Eins der Content Management Systeme in Europa betrachtet. Ihre Unterstützung von mehreren Programmiersprachen für die Schnittstelle und umfassende Funktionalität macht die Anwendung zu einer optimalen Lösung für die Verwaltung von dynamischen inhaltsgesteuerten Websites.

Die Firmen-Website (`http://www.six.de`) und der SixCMS-spezifische URL (`http://www.sixcms.com`) liefern zusätzliche Informationen über die Firma, ihre Kunden und die Lösungen. Diese Websites sind zwar ein hervorragendes Medium, um Informationen zur über die Firma und ihre Lösungen zu verbreiten, aber Mund-zu-Mund-Propaganda von zufriedenen Kunden begründen ebenso den Erfolg und den Zustrom neuer Kunden. Die Präsentation im Web ist auch eine Gelegenheit, um SixCMS in Aktion zu zeigen und kundenspezifische Demos und Test-Websites zu erstellen.

8.2.3 Warum PHP?

Benutzerfreundlichkeit und geringe Lizenzgebühren waren für Six natürlich wesentliche Kriterien für die Wahl von PHP. Kosten und Lizenzanforderungen sind bei PHP kein Thema. Bei anderen Optionen können leicht Kosten in Höhe von Tausenden von Dollars pro Entwicklung oder aktivem Server entstehen. Anbieter, welche die Open-Source-Technologie verwenden, haben damit einen beträchtlichen Vorteil. Sie kommen nicht in die Verlegenheit, mit

dem Kunden komplizierte abgestufte Lizenzgebühren zu erörtern, sondern können mit ihm die Kosten anhand der gewünschten Funktionalität der Anwendung darlegen. Ein weiterer Aspekt war die Leistung von PHP. Die Möglichkeit, PHP als Modul auf dem Apache-Webserver zu integrieren, erbrachte Leistungsvorteile gegenüber anderen CGI-Methoden. Aus Firmensicht war es einfach vernünftig, PHP zu benutzen, da es kosteneffektiv ist und den Einsatz von leistungsfähigen Anwendungen ermöglicht. Zusätzlich haben die Six-Entwickler für einige Bereiche Hexbase, WebSQL, WebObjects, Cold Fusion, Oracle Web Application Server, CGI, embperl, Java, C und Perl unter die Lupe genommen und getestet. PHP ist zwar nicht die einzige Entwicklungsplattform, die bei Six verwendet wird (in einigen Bereichen wird immer noch Perl verwendet), aber bereits über 90% der Entwicklungen wurden hier mit PHP erstellt. Eine Entscheidung, die Six auch heute noch unterstützt. Alle PHP-Anwendungen wurden zuerst mit PHP konzipiert und entwickelt.

Zur Entscheidung trugen auch die Open-Source-Aspekte von PHP bei. In einer Umgebung, in der Linux, Apache und MySQL verwendet werden, fügt sich PHP sehr gut ein. Durch die weitverbreitete Verwendung und die wachsende Akzeptanz des Open-Source-Systemansatzes, kann Six diese Umgebung sehr effektiv in seinen kommerziellen Umgebungen einsetzen, wie etwa SixCMS. Dies ist ein sehr guter Ansatz; er bietet eine exzellente Methode, um wertvolles geistiges Kapital auf kosteneffektive Weise zu erhalten. Seit einiger Zeit bekommen Open-Source-Technologien mehr und mehr Unterstützung aus der Industrie, insbesondere da ihre Verwendung immer weiter verbreitet wird und immer mehr Lösungen angeboten werden, die auf dieser Technologie basieren. Eine der Vorteile bei Open-Source-Lösungen ist, dass Sie zumindest selbst einen Blick auf den Quellcode werfen können, wenn etwas nicht so funktioniert wie erwartet. Six hat außerdem die Erfahrung gemacht, dass es einfacher und schneller geht, Unterstützung, Änderungen und Fehlerkorrekturen für Open-Source-Elemente zu bekommen als für kommerzielle Komponenten. Eine einfache Anfrage bei einer der Newsgroups oder Mailinglisten bringt häufig einige qualifizierte und hilfreiche Antworten – in wenigen Stunden, wenn nicht gar Minuten. Kommerzielle und propriätere Anwendungen und Umgebungen sind kompliziert, und häufig in Hinsicht auf Kosten und propriätere Mechanismen alles andere als optimal.

Inzwischen unterstützt Six die Entwicklung von PHP sowohl direkt als auch indirekt. Ein Angestellter von Six, Dr. Egon Schmid, gehört zur PHP-Entwicklungsgruppe und ist Mitverfasser des PHP-Buchs »PHP. Dynamische Webauftritte professionell realisieren« (Markt u. Technik, München 1999). Zusätzlich geben Six-Entwickler regelmäßig Empfehlungen und Meinungen in den PHP-Mailinglisten und -Usergroups. Dieser Aufwand wird sowohl in als auch außerhalb der Arbeitszeit unterstützt und gefördert.

8.2.4 Technologieüberlegungen

Six empfiehlt in vielen Situationen die Verwendung von PHP, und verwendet es selbst für die meisten seiner Anwendungen, einschließlich SixCMS. Insbesondere für layoutorientierte Websites mit geringem oder mittlerem Datenverkehr, bzw. in Situationen, in denen eine Anwendung schnell von Grund auf erstellt werden muss, bietet PHP eine exzellente Plattform. Natürlich hängt alles vom Umfang des Projekts ab. Wie bei jeder anderen Plattform hat auch PHP Beschränkungen. In sehr großen Projekten oder auf sehr großen Websites mit extrem vielen Aktivitäten, kann PHP ein Problem darstellen, insbesondere wenn die Laufzeit des Skript in Betracht gezogen wird. Wenn Leistungsfragen in Bezug auf die Belastung des Systems durch PHP auftauchen, sollten Caching und die Vorverarbeitung von Programmcode eingeführt werden, um statische Seiten zu erzeugen. Ganz allgemein (dies hängt von einer Reihe von Faktoren ab) empfiehlt es sich, statische Seiten zu erzeugen, wenn eine Website etwa 500.000 volldynamische Seitenansichten pro Monat hat, bzw. fünf bis zehn Abfragen gleichzeitig für eine Seite.

In bestimmten Situationen verwendete Six auch Mod_Perl, aber die Serverkonfiguration und der Bedarf an Kerneltuning rückten bei dieser Methode immer mehr in den Vordergrund. Perl ist sehr gut durchdacht und wird bereits seit einiger Zeit verwendet. Daher hat es ein sehr konsistentes Sprachschema und eine einheitliche Syntax. Da Mod_Perl außerdem kompiliert ist und im Speicher bleibt, kann eine Website von unterschiedlichen Komponenten erzeugt werden, die ihre eigenen benannten Speicher haben und unabhängig voneinander kompiliert werden können. Mit Mod_Perl können Sie direkt in die Apache-Bearbeitungsroutine eingreifen. Damit wird es zu einer sehr schönen Ergänzung für PHP und andere Skriptsprachen, zur Implementierung von Proxies, für spezifische Neuschreibregeln und Posttransaktionsroutinen u.a. Des weiteren hat Six Java auf der Server- und der Clientseite getestet. Obwohl es in Bezug auf die Leistung und die Kompatibilität einige Probleme gab, scheint das serverseitige Java für bestimmte zukünftige Anwendungen vielversprechend. Im Vergleich zu anderen Technologien, wie etwa Mod_Perl, Java, Cold Fusion und ASP, ist PHP im Allgemeinen günstiger, bietet eine hohe Leistung, integriert sich gut in andere Technologien und arbeitet auf dem Apache-Server sehr stabil. Gibt es auch Probleme? Im Grunde genommen ist die PHP-Plattform recht gut, Entwickler sollten jedoch die Grenzen kennen und die manchmal vorkommenden leichten Änderungen in der Programmiersprache oder den Funktionsaufrufen verschiedener Version beachten. Zusätzlich sollten Probleme, wie die Speicherbehandlung, insbesondere bei Zeichenketten die mehrere Megabytes des Serverspeichers belegen (ein Problem, dessen sich PHP 4.0 hoffentlich annimmt), und die schlechte Oracle-Integration, berücksichtigt werden. Weitere Defizite, welche die Six-Entwickler notierten, sind die fehlende Konsistenz der Sprachsyntax und des Design sowie implizite Typumwandlung.

Ein weiteres Problem bei PHP ist die Tatsache, dass Sie den gesamten Quellcode Ihrer Anwendung an den Kunden oder Partner weiterleiten müssen. Der Wert Ihrer Anwendungen, in Form von geistigem Eigentum und Kapital, und die Notwendigkeit, diese zu schützen, können für Sie wesentlich sein. Six hat für den Entwurf der Website mit einer Reihe von Werbe- und Designagenturen zusammengearbeitet, die begannen, sich neben ihrem grafischen Fähigkeiten auch Programmierkenntnisse zuzulegen, nachdem sie offenen Zugriff auf den Six-Code hatten, um sich als Mitbewerber zu behaupten. Dies sind Firmen, die versuchen Komplettlösungen für das Web anzubieten, im Gegensatz zu Six, die sich auf ihre Kernkompetenz der Anwendungsentwicklung konzentrieren. Man kann sicherlich argumentieren, dass sich die Technologie dermaßen schnell entwickelt, dass das Problem, dass jedermann Ihren Code sehen kann, langfristig kein wesentliches Problem darstellt. Die Möglichkeit mit PHP einen Bytecode-Compiler zu verwenden, sobald dieser verfügbar ist, wird die Situation erheblich verbessern. Six schließt derzeit wichtige Funktionen in der Datenbank selbst ein, um die Zugänglichkeit des PHP-Codes und lokale Änderungen durch Dritte zu begrenzen.

8.2.5 PHP im Einsatz

Die beiden zuvor genannten kommerziellen Produkte SixCMS und SixAIM sind Beispiele von Six-Anwendungen, die mit PHP 3.0 realisiert wurden. In der Version 3.0 hat SixCMS 283 Dateien, die 25.000 Programmzeilen enthalten und 1,2 MB Speicher belegen. Die Anwendung verwendet eine MySQL-Datenbank, die aus 40 Tabellen besteht. Im Vergleich dazu hat SixAIM etwa 71.000 Programmzeilen im PHP 3.0-Code (sowie weitere 10.000 Programmzeilen, die in Oracles PL / SQL-Sprachcode geschrieben sind) und belegt 2 MB Speicher. Die Oracle-Datenbank umfasst nahezu 200 Objekte: 65 Tabellen, 25 Datenbankfunktionen, 15 Prozeduren und 8 Pakete, die wiederum zusätzliche 45 Funktionen und 30 Prozeduren enthalten (nicht gezählt spezielle kundenspezifische Funktionen). Six hat die Standard- oder Basisversion von PHP verwendet und verwendet sie immer noch, insbesondere die Versionen 3.0.6, 3.0.7 und 3.0.12 sowie zu Testzwecken 4.0b3. Six plant für neue Projekte auf die Version 4.0 umzusteigen, sobald diese keine Beta-Version mehr ist und stabil läuft. Six hat zwar eine Reihe von Lösungen von der PHP-Version 2.x nach 3.x migriert, aber dies ist eine zeitaufwendige Aufgabe, bei der sich auch Programmierfehler einschleichen können, selbst wenn die Migration mit Skripten automatisiert wird. Für Syntax- und Sprachänderungen sowie funktionale Variationen, wie etwa bei der Handhabung von Arrays und der Deklaration von Variablen, werden zusätzliche Migrationszeiten und Testanforderungen benötigt.

PHP hat sich bei Six und seinen Kunden gut bewährt, ist aber, wie bereits erwähnt, nicht unbedingt ideal für Websites mit höherem Datenverkehr. Einige Caching-Konzepte waren hilfreich und außerdem verwendete Six eine

Reverse-Proxy-Technik, so dass PHP-fähige Server nicht auf die Bereitstellung von Grafiken und statischen Seiten beschränkt sind. Six verwendet zur Sitzungsverwaltung eigene, selbstentwickelte Mechanismen, und setzt häufig Cookies für sitzungsabhängige Parameter ein. Das Unternehmen hat bisher keinen großen Gebrauch von PHPLib gemacht, aber hat sich von einigen exzellenten Funktionen und Ideen daraus inspirieren lassen. Bei Bedarf wurde XML in eine Reihe von kundenspezifischen Anwendungen integriert. In verschiedenen Projekten wurden zusätzliche offene Standards, wie etwa LDAP, SMTP und X509 eingesetzt. In einigen seiner Entwicklungsprojekte hat Six kürzlich mit CVS eine Versionskontrolle eingeführt. Dies soll in Zukunft auf alle Entwicklungsprojekte erweitert werden. Der Anstieg der Programmierer und die Eröffnung einer Filiale in Berlin erforderten ein Umdenken in Bezug auf die Organisation von verteilten Entwicklungsteams und Projekten. Hierfür hat Six auch eine Reihe von eigenen Werkzeugen (mit PHP) entwickelt, um die Verwaltung der Projekt- und Entwicklungsressourcen zu erleichtern. Einige Lösungen hat Six mit unterschiedlichen Datenbanken realisiert, einschließlich Oracle, MySQL, PostgresSQL, Microsoft SQL Server und Sybase. Aufgrund seiner Open-Source-Philosophie und Kostenüberlegungen bevorzugt Six für die meisten nicht transaktionsbasierten Anwendungen die Verwendung von MySQL und für transaktionsbasierte Anwendungen, wie SixAIM, Oracle.

Sowohl für Six als auch für seine Kunden hat PHP stets eine große Rolle gespielt und spielt sie noch. Die Wahl und der Einsatz von PHP wurde im Dezember 1996 mit der PHP-Version 2.0 beta getroffen, die heute noch eine zentrale Komponente in SixCMS ist. Die Verwendung von PHP hat nicht nur die Open-Source-Philiosphie der Firma vorangetrieben, sondern auch eine komfortable und angemessene Umgebung für die Entwicklung von Highend-Anwendungen bereitgestellt.

8.2.6 PHP: Ein Geschäftsvorteil

Viele der funktionellen Verbesserungen und Leistungsvorteile der Six-Anwendungen lassen sich direkt in PHP realisieren, genauso wie die Vorteile der raschen Entwicklung und die Möglichkeit, Änderungen zügig und ohne erneutes Kompilieren vorzunehmen. Weitere Vorteile sind das einfache Verständnis und die effektive Nutzung von PHP ohne vorherige Kenntnisse oder Erfahrungen. Diese Vorteile erhöhen die Leistung und reduzieren gleichzeitig die zusätzlichen Entwicklungskosten. Diese Branche diktiert sehr kurze Entwicklungs- und Implementierungszeiten, und PHP bietet eine zuverlässige Middleware oder Anwendungsserverkomponente, die diese Laufzeitanforderungen erfüllen.

8.3 MarketPlayer.com

Diese ausführliche Fallstudie über eine Firma, die PHP in der Praxis in einer Serverfarm mit hohem Datenverkehr nutzt, stammt von Ph.D. Eric B. Schorvitz und John E. Joganic von MarketPlayer.com.

8.3.1 Firmenhintergrund

MarketPlayer.com bietet ein reales Finanztraining für einzelne Investoren, die erlernen möchten, wie sie an der Börse Geld machen. MarketPlayer.coms Werkzeuge zur Börsenüberwachung und Diagrammerstellung, die Institutionsqualität haben, helfen Investoren ihre eigenen Investitionsstrategien zu entwickeln; und die risikolosen, spaßigen und aufregenden Börsenwettbewerbe ermöglichen Investoren, ein Investmentportfolio zu handeln und zu testen.

Die Produkte von MarketPlayer.com sind über die firmeneigene Website unter `www.marketplayer.com` frei verfügbar, außerdem über strategische inhaltsbezogene Allianzen mit führenden Medien, Service Providern und Finanzdienstleistern einschließlich AOL, CNBC, E*TRADE, money.com, CompuServe, ESEC (Pan-European-Partner), internet.com und anderen.

MarketPlayer.com beschäftigt 24 Mitarbeiter in den Bereichen Marketing, Produktentwicklung, Kundenservice und Programmierung. Drei der Programmierer besitzen fundierte Kenntnisse in C und PHP, zwei weitere sind begnadete PHP-Programmierer.

8.3.2 Die PHP Produkte

Die von MarketPlayer.com angebotene Produkte lassen sich in zwei grundlegende Kategorien unterteilen: quantitative Finanzanwendungen und Simulationsprogramme für den Online-Börsenhandel. Um sich in diesen Bereichen zu behaupten, muss sich MarketPlayer.com ständig entscheiden, wie diese Anwendungen und der Inhalt verbessert werden kann, um der Konkurrenz stets einen Schritt voraus zu sein. MarketPlayer.com ist die einzige Website, auf der Investoren Aktienkurse über eine Zwölf-Monatszeitraum verfolgen und grafisch darstellen können, um Erträge abzuschätzen. Eine Technik, die auch von institutionellen Finanzmanagern bevorzugt wird. MarketPlayer.com hat bereits mehrere Patente eingereicht, um die einzigartige Weise zu schützen, in der die Online-Börsenspiele funktionieren. Außerdem verfügt MarketPlayer.com über ein Expertenteam von Produktentwicklern und Programmierern, welche die Aufgabe haben, neue und nützliche webbasierte Finanzwerkzeuge zu entwerfen, die dem einzelnen Investor helfen, das notwendige Wissen und die Erfahrungen zu sammeln, um in der Gemeinschaft der Investoren erfolgreich zu sein.

8.3.3 Warum PHP?

MarketPlayer.com verwendet PHP, da dieses in der Lage ist, eine schnelle Entwicklungsumgebung bereitzustellen. Mithilfe von PHP kann MarketPlayer.com innerhalb von Tagen konzeptionelle Entwürfe direkt vom Reißbrett in einer Beta-Version umwandeln. Teile der Anwendung, die optimiert werden müssen (in Bezug auf die Geschwindigkeit), können dann herausgenommen, in C geschrieben und in Form von Funktionen in eine benutzerspezifische PHP-Bibliothek integriert werden.

MarketPlayer.com verwendete PHP erstmals im Oktober 1999, als das Unternehmen beschloss, die Produktentwicklungsgruppe zu erweitern. Zuvor setze MarketPlayer.com das SSI-Verfahren ein, um dynamische Seiten zu erstellen. Ein Großteil der Firma verwendete also zur Webentwicklung ganz bequem WYSIWYG-Anwendungen. PHP ermöglichte MarketPlayer.com, anspruchsvolle dynamische Websites zu erstellen und gleichzeitig die Begabungen der existierenden Gruppe zu nutzen, die den WYSIWYG-Ansatz für die Webentwicklung verwendete. Vor PHP hatte die Firme bereits Lösungen mit Perl und Java realisiert, doch nach der Einführung von PHP blieben diese anderen Sprachen bald auf der Strecke.

Die Tatsache, dass PHP ein Open-Source-Programm ist, bietet für MarketPlayer.com gegenüber anderen Skriptsprachen viele Vorteile:

- Anwendungen können leicht optimiert werden, indem zum PHP-Quellcode Module und Funktionen hinzugefügt werden, die MarketPlayer.com-spezifisch sind.
- Die Unterstützung für diese Technologie wächst täglich.
- Viele Einzelpersonen stellen der Gemeinschaft erstklassige PHP-Anwendungen zur Verfügung, und die GNU-Lizensierung beschleunigt die Entwicklung von Schlüsselprodukten erheblich.

8.3.4 Vorteile von PHP für die Produktentwicklung von MarketPlayer.com

MarketPlayer.com zieht PHP anderen Technologien nicht nur wegen des einfachen Designs vor, sondern auch wegen der Verfügbarkeit und Erweiterbarkeit des Quellcodes, der guten Integration auf dem Apache-Webserver und vor allem seiner Ähnlichkeit zur Programmiersprache C. Die zweite Wahl wäre möglicherweise Perl gewesen, aber mit dieser waren die MarketPlayer.com-Programmierern nicht vertraut, und Erweiterungen für Datenbanken und proprietäre Algorithmen wären sehr kostspielig gewesen. Cold Fusion wurde nicht berücksichtigt, und durch die enge Verknüpfung zu Windows-Plattformen wurde ASP von vornherein ausgeschlossen. Bis vor kurzem stützte sich MarketPlayer.com auf serverseitiges Java für spezifischen,

vom Benutzer zu konfigurierende, Anfragen. Mit der Zeit wurde der Code zu umfangreich, um ihn zu pflegen, und wurde durch eine Lösung ersetzt, die komplett in PHP und C geschrieben ist. Die derzeit laufende Beta-Version ist fünfmal schneller als das Java-Produktionssystem und hat noch viel Raum für Optimierungen.

Im System wurden keine fundamentalen Mängel gefunden, doch die relativen Pfade in der `include()`-Anweisung führten zu einiger Frustration, insbesondere bei verschachtelten `includes`. Die C-Quell- und Headerdateien verwenden als Basis für die relativen Pfade ihr eigenes Verzeichnis, während die `include`-Anweisung von PHP das Verzeichnis der ursprünglich aufgerufenen Datei verwendet. Ein eigenes Verzeichnis für `includes` zu pflegen und diese Dateien aus einer hierarchischen Verzeichnisstruktur heraus zu referenzieren, kann kompliziert werden. Deshalb entschloss man sich, eine Variable für das `include`-Verzeichnis zu erzeugen, und bei der Einbindung der Headerdateien nur absolute Pfade zu verwenden.

In PHP 3.0 war es außerdem schwierig, benutzerspezifische Erweiterungen zu integrieren. MarketPlayer.com hat PHP 4.0 bereits getestet und ist mit dieser Version sehr viel glücklicher. Aber solange die Betatestphase noch nicht abgeschlossen ist, ist eine Einsatz auf Produktionsservern noch nicht zu empfehlen. Derzeit verwendet MarketPlayer.com die Version 3.0.12.

Bei Betrachtung aller Aspekte konnte MarketPlayer.com keine Beschränkung durch fehlende Komponenten in PHP feststellen. Es besteht sicherlich Interesse an einem Tool zur Fehlersuche, das PHP bedient, sofern dies jemand entwickelt. In unmittelbarer Zukunft wird jedoch eher ein einfacherer Mechanismus zur Erweiterung der Funktionalität benötigt, bei dem kein Overhead durch einen `dl()`-Aufruf entsteht oder der PHP-Quellcode direkt modifiziert werden muss.

8.3.5 Bewährung im Alltag

Die Anwendungen von MarketPlayer.com reichen von einzelnen Skripten bis hin zu kompletten Websiteverwaltungspaketen. Insgesamt gibt es derzeit mindestens 10.000 Programmzeilen in PHP und für das nächste Quartal sind weitere 40.000 geplant. Als Teil des Entwicklungsprozesses wurde übliche Funktionen zu eingebetteten Funktionen, die in C realisiert wurden und deren C-Code schließlich in Bibliotheken integriert wurde, die von niedrigeren Integratoren genutzt werden können. Diese Strategie der Top-Down-Programmierung garantiert, dass der Fokus nicht auf der Codeproduktion, sondern auf der Produktentwicklung liegt, was im Bereich der Entwicklung von Webinhalten unerlässlich ist.

PHP hat sich als skalierbar und schnell erwiesen, da es von MarketPlayer.com auf Websites mit hohem Datenverkehr ohne Störungen verwendet werden

kann. Bei Beschränkungen wäre eine Migration auf schnellere Maschinen oder eine Änderung der Netzwerktopographie empfehlenswerter als der Wechsel auf eine andere Technologie. Durch permanentes Profiling können Engpässe gefunden werden. Es erleichtert auch die Entscheidung, welcher PHP-Code in C neu geschrieben und eingebunden werden muss.

8.3.6 Sitzungsverwaltung

MarketPlayer.com verwendet zwei Techniken zur Verwaltung von Sitzungen. Im ersten Fall werden die Benutzerdaten verschlüsselt und auf dem Clientrechner in Form eines Cookie gespeichert. Dies hat den Nachteil, dass jeder Server, der dynamische Inhalte bereitstellt, den Cookie entschlüsseln können muss. In der Entwicklungsumgebung wird die Entwicklung neuer Funktionen so lange verzögert, bis die Speicherung mittels Cookies realisiert ist.

Der zweite und portierbarere Fall schließt eine Datenbanktabelle ein, die beliebige einzigartige Sitzungskennnummern mit authentisierten Benutzern abgleicht. Sobald der Benutzer über einen gültigen Namen und ein Passwort authentisiert ist, wird eine neue Sitzungskennnummer generiert und zusammen mit der IP-Adresse, der Zeit und dem Benutzernamen in der Benutzertabelle gespeichert. Der Benutzer braucht nur eine gültige Sitzungskennnummer und eine Verbindung, über die ursprünglich authentisierte IP-Adresse, um eine Verbindung herzustellen. Vom Standpunkt der Sicherheit kann eine Sitzungskennnummer für ungültig erklärt werden oder ablaufen, indem sie aus der Datenbank entfernt wird. Die Verwendung der Sitzungskennnummer von einer anderen IP-Adresse aus wird stillschweigend ignoriert. Benutzerdaten werden nur einmal gesendet. In Hinsicht auf die Leistung muss ein Datenbankaufruf nur einmal getätigt werden und alle Benutzerinformationen können über die Sitzungskennnummer durch Verknüpfen von Tabellen gesammelt werden.

Die Sitzungskennnummer wird dem Benutzer auf zwei Arten zur Verfügung gestellt. Im ersten Fall wird sie in alle URLs auf der Seite eingefügt, so dass die Information in allen Links auf Seiten derselben Website enthalten ist. URLs werden mithilfe einer Funktion generiert, die überprüft, ob der Benutzer authentisiert ist oder nicht, und das richtige HTML ausgibt. Außerdem wird sie als Cookie an den Benutzer geschickt. Dies verhindert, dass falsch definierte Links dazu führen, dass der Client die Kennnummer vergisst. Viel wichtiger ist, dass Browser, die Cookies nicht unterstützen oder gar verwerfen, sich trotzdem auf der Website anmelden können. Cookie-Variablen überschreiben `Get`-Variablen. Wenn also der Benutzer ein Lesezeichen auf eine Seite setzt, überschreibt die aktuelle Sitzung jegliche Kennnummer, die in dem URL im Lesezeichen angegeben ist. Wenn der Benutzer die Website schließlich verlässt und zurückkehrt, bevor die Sitzungskennnummer ausläuft, bleibt er authentisiert.

8.3.7 PHP-Serverintegration

Da die PHP-Bibliothek von MarketPlayer.com permanent verändert wird, wurde der Apache-Server für ladbare Module kompiliert, und eine Shared-Object-Implementierung für PHP erzeugt. XML wird nicht für die Inhalte, sondern nur für interne Konfigurationsdateien verwendet. Die GNU-Mechanismen `autoconf` und `configure` werden auf verschiedenen Plattformen für Entwicklungs- und Produktionstools eingesetzt. Nahezu der gesamte MarketPlayer.com-Quellcode ist entweder herkömmlicher proprietärer Code oder Open-Source-Code. Die Datenbankinfrastruktur setzt sich zusammen aus MySQL, die für die Prototypenerstellung und kleinere Aufgaben verwendet wird, und Velocis für die Kernfunktionen. Die Effizienz von Velocis gegenüber anderen Datenbanklösungen muss sich noch zeigen. Bisher scheint sich jedoch bei dem vorhandenen Programmcode das Netzwerkmodell gegenüber relationalen Varianten zu behaupten. Dies ändert sich vielleicht, wenn die Migration auf die neue Technologie fortgeschritten ist.

8.3.8 Codeverwaltung

Die Entwickler, die bei MarketPlayer. für den Inhalt der Content Management Systeme zuständig sind, arbeiten derzeit unter Windows NT mit der Versionskontrolle Source Safe. Mit der neuen Technologie wird der gesamte Code jedoch unter CVS gespeichert. Sobald die Redaktionsabteilung die notwendigen Tools hat, um unter Linux arbeiten zu können, soll der Code wieder in CVS zurückgespeichert werden.

Produktmanager stellen Entwicklungsteams zusammen, legen das Design fest, entwerfen die Codestruktur und überwachen die Implementierung durch die Teammitglieder. Anschließend wird eine Prototypversion entwickelt und als Alphaversion freigegeben und in das CVS eingepflegt. Diese Komponenten werden anschließend an die anderen Entwickler verteilt und durch die Produktmanager auf Sicherheit, Effizienz und Design überprüft. Nachfolgende Versionen unterscheiden sich durch hinzugefügte Funktionen, Fehlerbereinigungen und Verbesserung des Design. Designteams sind spezialisiert und auf bestimmte Aufgaben ausgerichtet. Damit können sie schnell auf die Bedürfnisse des Unternehmens reagieren.

Während der Entwicklung der Prototypversion können die Schnittstellen der Elemente in PHP noch relativ vage sein, da die Funktionalität normalerweise direkt im Code enthalten ist. Sobald die Module zusammengefügt werden, kann die Parameterordnung und die Funktionsnomenklatur festgelegt werden, ohne dass der Code massiv umgeschrieben werden muss. Ausstehende Funktionen können in PHP leicht simuliert oder in der Entwicklungsumgebung umgangen werden. Firmenexterne Funktionen fügen sich nahtlos in PHP ein. Dies begrenzt das Design auf den Zeitraum der Prototypentwicklung und verhindert, dass sich Verzögerungen auf andere Entwickler auswirken.

8.3.9 Die Zukunft

MarketPlayer.com entwickelt sich derzeit in Richtung auf eine 100%ige Website. Mit über acht Millionen Seitenansichten pro Monat auf allen seinen Websites hat MarketPlayer.com keine Probleme in der Handhabung des Datenverkehrs festgestellt und ist der Überzeugung, dass PHP jede Datenfrequenz bewältigt.

8.4 Zusammenfassung

Wie Sie feststellen konnten, unterscheidet sich das heutige PHP stark von seinen Ursprüngen. Obwohl es einen enormen Einfluss auf die Open-Source-Gemeinschaft hatte, wurde PHP zu Beginn noch als Amateurlösung betrachtet und als nettes Spielzeug abgetan. Natürlich waren die »Pretty Hip People« (wie sich die PHP-Benutzer manchmal selbst bezeichnen) von ihrem eigenen Tool vollkommen überzeugt, mussten aber zugeben, dass es sich manchmal im Wettbewerb nicht messen konnte. Im Laufe der Entwicklung der Version 3 wagte PHP den Angriff auf alte Giganten, wie ASP, Cold Fusion u.a – und gewann den Kampf immer häufiger.

In diesem Kapitel haben wir Ihnen drei Beispiele gezeigt, in denen PHP seine Konkurrenten erfolgreich aus dem Ring geworfen hat und bewies, dass es in der Praxis durchaus als leistungsfähiges Werkzeug taugt. Angesichts der schnell wachsenden Anzahl von Servern, auf denen PHP inzwischen läuft, und des riesigen Schritts, den PHP von der Version 3 zur Version 4 gemacht hat, glauben wir, dass PHP nun das erreicht hat, auf was es abzielte: Es ist ein hervorragendes Werkzeug zur raschen Entwicklung von stabilen und schnellen Skriptlösungen geworden.

8.5 Referenzen

[1] Die Umfrage finden Sie unter `www.netcraft.com/survey`.

Teil 3: Über die Grenzen von PHP hinaus

Pearson
Education

ADDISON-WESLEY

9 Erweiterung von PHP 4.0: Knacken wir den PHP-Kern

Jene, die wissen, reden nicht.
Jene, die reden, wissen nicht.

Manchmal ist PHP »wie vorliegend« einfach nicht genug. Für den durchschnittlichen Benutzer wird dieser Fall wohl eher selten auftreten, aber professionelle Anwendungen bringen PHP an die Grenzen seiner Leistungsfähigkeit in Bezug auf Geschwindigkeit oder Funktionalität. Aufgrund von Sprachbeschränkungen und Unverträglichkeiten, die auftreten, wenn Sie mit einer sehr großen Bibliothek aus Standardcode arbeiten, die an jedes einzelne Skript angehängt wird, können neue Funktionen nicht immer direkt implementiert werden. Deshalb muss eine andere Methode gefunden werden, um diese potentielle Mängel in PHP zu beseitigen.

Sobald dieser Punkt erreicht ist, ist es Zeit, das Herz von PHP unter die Lupe zu nehmen, und sich den Kern, d.h. den C-Code anzusehen, der PHP zum Laufen bringt.

> **Hinweis:** Dieses Kapitel behandelt nur die Erweiterung von PHP 4.0. Ein Großteil der Informationen ist zwar auch für PHP 3.0 relevant, trotzdem sind die Beispiele nicht darauf ausgelegt, zu PHP 3.0 kompatibel zu sein. Wenn sich jemand die Mühe macht, PHP zu erweitern, wird er unserer Meinung nach auf PHP 4.0 zurückgreifen. Angesichts der Vorteile der neuen PHP-Version ist eine Neukompilierung von PHP 3.0-Servern nicht sinnvoll.

Zum Zeitpunkt der Bucherstellung waren einige Funktionen in PHP 4.0 noch nicht fertiggestellt oder funktionsuntüchtig (einer der Hauptaspekte ist die thread-sichere Version von Zend).

9.1 Überblick

»PHP erweitern« ist einfacher gesagt als getan. PHP hat sich zu einem vollausgestatteten Werkzeug entwickelt, das aus ein paar Megabyte Quellcode besteht. Um ein System wie dieses zu knacken, müssen Sie einiges wissen und berücksichtigen. Bei der Strukturierung dieses Kapitels entschlossen wir uns schließlich für den Ansatz »Lernen durch praktische Anwendung«. Dies ist zwar nicht der wissenschaftlichste und professionellste Ansatz, aber die Methode, die am meisten motiviert und die besten Endresultate erzielt. In den folgenden Abschnitten erfahren Sie schnell, wie Sie die einfachsten Erweiterungen erzeugen und sofort einsetzen können. Danach gehen wir auf die fortgeschrittene

API-Funktionalität von Zend ein. Alternativ könnten wir auch versuchen, die Funktionalität, das Design, Tipps, Tricks usw. als Ganzes zu erläutern, um Ihnen einen Gesamteindruck zu verschaffen, bevor wir Sie in die Praxis entlassen. Dies ist zwar das »bessere« Verfahren, da es keine groben Häckereien erfordert, doch es kann auch sehr frustrieren und außerdem sehr energie- und zeitaufwendig sein. Deshalb haben wir uns für den direkten Ansatz entschieden.

Auch wenn dieses Kapitel versucht, soviel Wissen wie möglich über die innere Funktionsweise von PHP zu vermitteln, sollten Sie bedenken, dass es unmöglich ist, einen vollständigen Leitfaden für die Erweiterung von PHP zu geben, der in allen Fällen immer zu 100% funktioniert. PHP ist ein solch großes und komplexes Paket, dass seine innere Funktionsweise nur dann verstanden werden kann, wenn Sie sich mit ihm durch praktische Erfahrung vertraut machen. Deshalb möchten wir Sie ermutigen, mit dem Quellcode zu arbeiten.

9.2 Was ist Zend und was ist PHP?

Der Name *Zend* bezieht sich auf die Sprachmaschine, den Kern von PHP. Der Begriff *PHP* bezieht sich auf das komplette System, wie es sich nach außen darstellt. Dies mag zunächst etwas verwirrend erscheinen, aber es ist gar nicht so kompliziert (siehe Abbildung 9.1). Um einen Webskriptinterpreter zu implementieren, benötigen Sie drei Teile:

- den Interpreter-Teil, der den Eingabecode analysiert, übersetzt und ausführt,
- den Funktionalitäten-Teil, der für die Funktionalität der Sprache (seine Funktionen usw.) zuständig ist und
- den Schnittstellen-Teil, der mit dem Webserver spricht, usw.

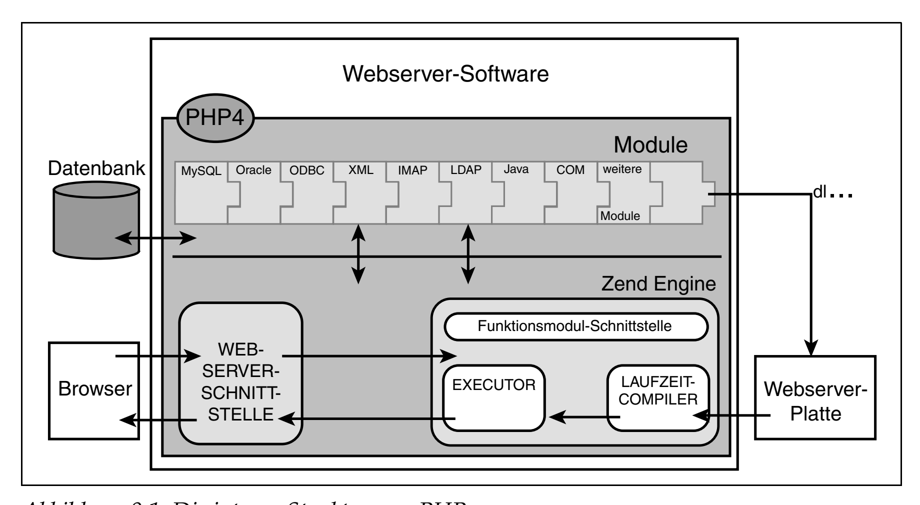

Abbildung 9.1: Die interne Struktur von PHP

Zend übernimmt den kompletten Teil 1 und einen Teil von Teil 2, PHP umfasst die Teile 2 und 3. Zusammen bilden sie das vollständige PHP-Paket. Zend selbst bildet nur den Sprachkern, der PHP mithilfe von einigen vordefinierten Funktionen in seiner grundlegendsten Form realisiert. PHP enthält alle Module, welche die hervorragenden Fähigkeiten der Sprache realisiert.

Die folgenden Abschnitte beschreiben, an welchen Stellen PHP erweitert werden kann und wie dies geschieht.

9.3 Erweiterungsmöglichkeiten

Wie in Abbildung 9.1 gezeigt, kann PHP hauptsächlich an drei Punkten erweitert werden: an den externen Modulen, den eingebauten Modulen und an der Zend Engine. Diese Option werden in den folgenden Abschnitten erörtert.

9.3.1 Externe Module

Externe Module können mithilfe der Funktion `dl()` zur Laufzeit des Skripts geladen werden. Diese Funktion lädt ein gemeinsam genutztes Objekt (Shared Object) von der Festplatte und stellt seine Funktionalität dem Skript zur Verfügung, an das es gebunden ist. Nachdem das Skript beendet ist, wird das externe Modul aus dem Speicher entfernt. Diese Methode hat sowohl Vorals auch Nachteile, wie die folgende Tabelle zeigt:

Vorteile	Nachteile
Externe Module erfordern kein erneutes Kompilieren von PHP.	Die Shared Objects müssen jedes Mal geladen werden, wenn ein Skript ausgeführt wird (bei jedem Treffer), was die Ausführung verlangsamt.
Durch Auslagern bestimmter Funktionen bleibt PHP relativ klein.	Zusätzliche externe Dateien füllen die Festplatte.
	Jedes Skript, das die Funktionalität einer externen Funktion nutzen möchte, muss einen Aufruf für `dl()` enthalten, oder das Tag `extension` in `php.ini` muss modifiziert werden (was nicht immer eine gute Lösung ist).

Alles in allem eignen sich externe Module hervorragend für die Produkte Dritter, für kleine Zusätze zu PHP, die selten benutzt werden, oder für Testzwecke. Um zusätzliche Funktionen schnell zu entwickeln, liefern externe Module die besten Ergebnisse. Bei häufiger Nutzung, größeren Anwendungen oder komplexeren Codes überwiegen jedoch die Nachteile.

Dritte ziehen möglicherweise in Betracht, das Tag `extension` in `php.ini` zu nutzen, um in PHP zusätzliche externe Module zu erzeugen. Diese externen Module werden vollständig vom Hauptpaket abgetrennt, was in kommerziellen Umgebungen recht praktisch ist. Damit können kommerzielle Distributoren Disketten oder Archive verschicken, die nur die zusätzlichen Module enthalten, und müssen sich nicht damit befassen, statische und in sich geschlossene PHP-Binärdateien zu erzeugen, die verhindern, dass andere Module an sie gebunden werden.

9.3.2 Eingebaute Module

Eingebaute Module werden direkt in PHP übersetzt und mit jedem PHP-Prozess mitgeführt. Ihre Funktionalität ist für jedes Skript, das ausgeführt wird, sofort verfügbar. Genau wie externe Module haben eingebaute Module Vor- und Nachteile, wie die folgende Tabelle beschreibt:

Vorteile	Nachteile
Das Modul muss nicht speziell geladen werden, die Funktionalität ist sofort verfügbar.	Bei Änderungen von eingebauten Modulen ist ein Neukompilieren von PHP erforderlich.
Es gibt keine externen Dateien, welche die Festplatte füllen, alles wird in den PHP-Binärdateien abgelegt.	Die PHP-Binärdateien wachsen und belegen zunehmend mehr Speicher.

Eingebaute Module eignen sich am besten, wenn Sie eine robuste Bibliothek mit Funktionen verwenden, die relativ unverändert bleibt, die eine mehr als durchschnittliche Leistung erfordert oder häufig von vielen Skripten auf Ihrer Website verwendet wird. Die Notwendigkeit, PHP neu zu kompilieren, wird schnell durch den Gewinn an Geschwindigkeit und seine Benutzerfreundlichkeit wettgemacht. Eingebaute Module sind jedoch nicht ideal, wenn Sie rasch kleine Zusätze entwickeln möchten.

9.3.3 Die Zend Engine

Natürlich können Erweiterungen auch direkt in der Zend Engine realisiert werden. Diese Strategie ist geeignet, wenn Sie eine Änderung im Sprachverhalten benötigen oder spezielle Funktionen brauchen, die direkt im Sprachkern integriert sind. Im Allgemeinen sollten Modifikationen an der Zend Engine jedoch vermieden werden. Änderungen führen hier zu Inkompatibilitäten mit anderen Programmen – und es wird kaum jemand Anpassungen an eine speziell zusammengestellte Zend Engine vornehmen. Da Modifikationen stets direkt im PHP-Quellcode durchgeführt werden, werden sie überschrie-

ben, sobald Sie ein Update mit den »offiziellen« Quellen machen. Daher wird diese Methode im Allgemeinen als schlechte Praxis betrachtet und aufgrund seiner Seltenheit in diesem Buch nicht behandelt.

9.4 Struktur der Quellen

Bevor wir uns direkt mit Fragen des Programmcodes beschäftigen, sollten Sie sich mit der Baumstruktur der Quellen vertraut machen, um sich schnell durch die PHP-Dateien durchzufinden. Dies ist unbedingt notwendig, um Erweiterungen einbinden und Fehler finden zu können.

Nach der Extrahierung des PHP-Archivs sehen Sie eine Verzeichnisstruktur, die jener in Abbildung 9.2 ähnelt.

- php-4
  - .libs
  - build
  - dl
  - ext
  - libs
  - pear
  - regex
  - sapi
  - scripts
  - tests
  - TSRM
  - win32
  - Zend

Abbildung 9.2: Grundlegende Baumstruktur der PHP-Quellen

Hinweis: Bevor Sie den Rest dieses Kapitels durcharbeiten, sollten Sie sich eine saubere, nicht veränderte Baumstruktur der Quellen von Ihrem bevorzugten Webserver besorgen. Wir arbeiten mit Apache (erhältlich unter `http://www.apache.org`) und natürlich mit PHP (erhältlich unter `http://www.php.net` – muss dies noch erwähnt werden?).

Alternativ können Sie die bereitgestellten Quellarchive von der CD-ROM verwenden, die mit diesem Buch mitgeliefert wird. Alle Beispiele in diesem Buch arbeiten mit den Quellarchiven von der CD-ROM. Dies können wir nicht für alle Versionen garantieren, die Sie vom Netz herunterladen. Da sich Open-Source-Software extrem rasch ändert, können bereits Änderungen in den Quelldateien vorgenommen worden sein, so dass die Versionen auf der CD-ROM möglicherweise veraltet sind und nicht alle Funktionen besitzen, die Sie brauchen. Wenn Sie die offiziellen Archive von den entsprechenden Websites

nicht zum Laufen bringen können, experimentieren Sie mit den Archiven auf der CD-ROM und versuchen Sie, von dort aus fortzufahren.

Stellen Sie sicher, dass Sie eine funktionierende PHP-Umgebung selbst kompilieren können! Auf diesen Aspekt werden wir hier nicht näher eingehen, da Sie über diese grundlegenden Kenntnisse bereits vor der Lektüre dieses Buches verfügen sollten.

Die folgende Tabelle beschreibt den Inhalt der wichtigsten Verzeichnisse.

Verzeichnis	Inhalt
`php-4`	Wichtigste PHP-Quelldateien und Headerdateien; hier finden Sie alle API-Definitionen, Makros usw. von PHP (wichtig).
`dl`	Verzeichnis für dynamisch ladbare Module; enthält die spezielle Headerdatei `phpdl.h` sowie die notwendigen Dateien für die Automatisierung des make-Prozesses. Dieses Verzeichnis ist ein Überbleibsel aus dem vorherigen PHP-Erstellungssystem und soll aus dem Programmverzeichnis entfernt werden. Von seiner Verwendung wird daher abgeraten.
`ext`	Verzeichnis für dynamische und eingebaute Module; standardmäßig sind dies die »offiziellen« PHP-Module, die in das Hauptprogrammverzeichnis integriert wurden. In PHP 4.0 können diese Standarderweiterungen als dynamisch ladbare Module kompiliert werden (zumindest jene, die diese Funktion unterstützen).
`pear`	Verzeichnis für die Ablage von PHP-Klassen. Zum Zeitpunkt der Bucherstellung befindet sich dieses Verzeichnis noch in der Konzeption, es soll aber etwas Ähnliches wie CPAN für Perl eingerichtet werden.
`sapi`	Enthält den Code für die verschiedenen Server-Abstraction-Layers.
`TSRM`	Verzeichnis für den Thread-Safe-Resource-Manager (TSRM) für Zend und PHP.
`Zend`	Verzeichnis für Zends Datei; hier finden Sie alle API-Definitionen, Makros usw. von Zend (wichtig).

Alle Dateien zu erörtern, die im PHP-Paket enthalten sind, würde den Rahmen dieses Kapitels sprengen. Sie sollten sich jedoch folgende Dateien näher ansehen:

- `php.h`, die sich im Hauptverzeichnis von PHP befindet. Die Datei enthält die meisten Marko- und API-Definitionen von PHP.
- `zend.h`, die sich im Hauptverzeichnis von Zend befindet. Die Datei enthält die meisten Makros und Definitionen von Zend.
- `zend_API.h`, die sich ebenfalls im Zend-Verzeichnis befindet, und die API von Zend definiert.

Daneben sollten Sie sich auch einige der Dateien betrachten, von denen diese Dateien abhängen; z.B. jene, die sich auf den Zend-Executor, die Unterstützung für PHP-Initialisierungsdateien und dergleichen beziehen. Nachdem Sie diese Dateien studiert haben, nehmen Sie sich die Zeit, ein wenig im Paket herumzuwandern, um sich einen Einblick in die gegenseitigen Abhängigkeiten aller Dateien und Module zu verschaffen, wie sie sich aufeinander beziehen und insbesondere wie sie einander nutzen. Dies hilft Ihnen auch, sich an den Programmierstil anzupassen, in dem die PHP-Quellen verfasst sind. Um PHP zu erweitern, sollten Sie diesen Stil schnell übernehmen.

9.4.1 Erweiterungskonventionen

Zend wird mithilfe einiger Konventionen erstellt; um seine Standards nicht zu verletzen, sollten Sie seine Regeln befolgen, die in den folgenden Abschnitten beschrieben werden.

9.4.2 Makros

Für fast jede wichtige Aufgabe liefert Zend vordefinierte Makros, die extrem praktisch sind. Die Tabellen und Abbildungen in den folgenden Abschnitten beschreiben die meisten der grundlegenden Funktionen, Strukturen und Makros. Die Makrodefinitionen sind hauptsächlich in `zend.h` und `zend_API.h` abgelegt. Wir schlagen vor, dass Sie sich mit diesen Dateien näher befassen, nachdem Sie dieses Kapitel gelesen haben. (Diese können Sie natürlich auch sofort lesen, aber es wird wahrscheinlich noch nicht alles für Sie sinnvoll erscheinen.)

9.4.3 Speicherverwaltung

Ressourcenverwaltung ist ein wesentlicher Faktor, insbesondere bei Serversoftware. Eine der wertvollsten Ressourcen ist der Speicher, und die Speicherverwaltung sollte mit besonderer Sorgfalt betrieben werden. Die Speicherverwaltung wurde teilweise in Zend eingebettet, und diese eingebetteten Funktionen sollten Sie konsequent nutzen, aus offensichtlichen Gründen: Sie ermöglichen Zend die vollständige Kontrolle über alle Speicherzuweisungen. Zend kann feststellen, ob ein Block in Gebrauch ist, unbenutzte Blöcke und Blöcke mit aufgehobenen Referenzen freigeben und damit Speicherlöcher vermeiden. Die folgende Tabelle zeigt die zu verwendenden Funktionen:

Funktion	Beschreibung
`emalloc()`	Ersetzt `malloc()`
`efree()`	Ersetzt `free()`
`estrdup()`	Ersetzt `strdup()`

Funktion	Beschreibung
estrndup()	Ersetzt strndup(). Schneller als estrdup() und binärsicher. Die Verwendung dieser Funktion wird empfohlen, wenn Sie die Länge des Strings kennen, bevor Sie ihn duplizieren.
ecalloc()	Ersetzt calloc()
erealloc()	Ersetzt realloc()

emalloc(), estrdup(), estrndup(), ecalloc() und erealloc() weisen internen Speicher zu; efree() setzt diese zuvor zugewiesenen Blöcke frei. Der von den e*() Funktionen verwaltete Speicher wird vom aktuellen Prozess als lokal betrachtet, und wird gelöscht, sobald das von diesem Prozess ausgeführte Skript beendet wird.

Warnung: Um residenten Speicher zuzuweisen, der nach der Beendigung des aktuellen Skripts erhalten bleibt, können Sie malloc() und free() verwenden. Dies sollten Sie jedoch nur mit extremer Vorsicht tun und nur, wenn die Zend-API dies erfordert. Ansonsten riskieren Sie Speicherlöcher.

Zend bietet außerdem einen thread-sicheren Ressourcemanager, der eine bessere und direkte Unterstützung für Webserver mit mehreren Teilprozessen bietet. Dafür müssen Sie allen Ihren globalen Variablen lokale Strukturen zuweisen, damit gleichzeitig mehrere Teilprozesse ausgeführt werden können. Da der thread-sichere Modus von Zend noch nicht fertiggestellt ist, können wir ihn in diesem Buch nicht behandeln.

9.4.4 Verzeichnis- und Dateifunktionen

Die folgenden Verzeichnis- und Dateifunktionen sollten in Zend-Modulen verwendet werden (sie verhalten sich genauso wie ihre C-Gegenstücke):

Zend-Funktion	Reguläre C-Funktion
V_GETCWD()	getcwd()
V_FOPEN()	fopen()
V_CHDIR()	chdir()
V_GETWD()	getwd()
V_CHDIR_FILE()	Hat als Argument einen Dateipfad und ändert das aktuelle Arbeitsverzeichnis auf das Verzeichnis der Datei.
V_STAT()	stat()
V_LSTAT()	lstat()

9.4.5 Handhabung von Strings

Strings werden von der Zend-Engine etwas anders gehandhabt als andere Werte, wie etwa Integer, boolesche Werte usw., die zur Speicherung ihrer Werte keine zusätzlichen Speicherzuweisungen benötigen. Wenn Ihre Funktion einen String zurückgeben soll, führen Sie eine neue Stringvariable in der Symboltabelle ein oder, um etwas Ähnliches zu tun, stellen Sie sicher, dass der Speicher, den der String belegen wird, zuvor mit den bereits genannten `e*()`-Funktionen zugewiesen wurde. (Dies erscheint Ihnen möglicherweise noch nicht sehr sinnvoll; behalten Sie es einfach im Hinterkopf, wir kommen in Kürze darauf zurück.)

9.4.6 Komplexe Typen

Komplexe Typen, wie etwa Arrays oder Objekte, erfordern eine andere Behandlung. Zend bietet eine separate API für diese Typen: sie werden über Hash-Tabellen gespeichert.

Hinweis: Um die Komplexität in den folgenden Programmbeispielen zu reduzieren, arbeiten wir zunächst nur mit einfachen Typen, wie Integer. Später werden wir die Erstellung anspruchvollerer Typen erörtern.

9.5 PHPs automatisches Erstellungssystem

PHP 4.0 verfügt über ein automatisches Erstellungssystem, das sehr flexibel ist. Alle Module befinden sich in einem Unterverzeichnis des Verzeichnisses `ext`. Neben seinen eigenen Quellen umfasst jedes Modul eine `M4`-Datei (siehe z.B. `www.gnu.org/manual/m4/html_mono/m4.html`) zur Konfiguration, und eine `Makefile.in`-Datei, die für die Kompilierung zuständig ist (die Ergebnisse von autoconf und automake; siehe z.B. `http://sourceware.cygnus.com/autoconf/autoconf.html` und `http://sourceware.cygnus.com/automake/automake.html`).

Beide Dateien werden zusammen mit `.cvsignore` automatisch von einem kleinen Shellskript namens `ext_skel` generiert, das sich im Verzeichnis `ext` befindet. Sein Argument gibt den Namen des Moduls an, das Sie erstellen möchten. Anschließend erzeugt das Shellskript ein Verzeichnis desselben Namens, zusammen mit den entsprechenden `config.m4`- und `Makefile.in`-Dateien.

Nach und nach läuft folgender Prozess ab:

```
root@dev:/usr/local/src/php4/ext > ./ext_skel my_module
Creating directory
Creating basic files: config.m4 Makefile.in .cvsignore [done].
```

```
To use your new extension, you will have to execute the following steps:

    $ cd ..
    $ ./buildconf
    $ ./configure                 (your extension is automatically enabled)
    $ vi ext/my_module/my_module.c
    $ make

Repeat the last two steps as often as necessary.
```

Diese Anweisung erzeugt die zuvor erwähnten Dateien. Um das neue Modul in den automatischen Konfigurations- und Erstellungsprozess zu integrieren, müssen Sie `buildconf` ausführen. Dadurch wird das Skript `configure` neu generiert, indem das Verzeichnis `ext` durchsucht und alle gefunden `config.m4`-Dateien eingebunden werden.

Bei der Ausführung von `configure` werden alle Konfigurationsoptionen analysiert und, basierend auf diesen Optionen und den Optionen, die Sie in `Makefile.in` angegeben haben, ein Makefile generiert.

Listing 9.1 zeigt die zuvor generierte Datei `Makefile.in`:

```
# $Id: Extending_Zend.xml,v 1.22 2000/05/22 20:02:58 till Exp $

LTLIBRARY_NAME        = libmy_module.la
LTLIBRARY_SOURCES     = my_module.c
LTLIBRARY_SHARED_NAME = my_module.la

include $(top_srcdir)/build/dynlib.mk
```

Listing 9.1: Die Standard-Makefile.in

Viel ist über sie nicht zu erzählen: Sie enthält die Namen der Eingabe- und Ausgabedateien. Sie könnten auch Erstellungsanweisungen für andere Dateien angeben, wenn sich Ihr Modul aus mehreren Quelldateien zusammensetzt.

Die Standard-`config.m4`-Datei, die Sie in Listing 9.2 sehen, ist etwas komplexer:

```
dnl $Id: Extending_Zend.xml,v 1.22 2000/05/22 20:02:58 till Exp $
dnl config.m4 for extension my_module
dnl don't forget to call PHP_EXTENSION(my_module)

dnl If your extension references something external, use with:

PHP_ARG_WITH(my_module, for my_module support,
dnl Make sure that the comment is aligned:
[  --with-my_module             Include my_module support])
```

```
dnl Otherwise use enable:

PHP_ARG_ENABLE(my_module, whether to enable my_module support,
dnl Make sure that the comment is aligned:
[  --enable-my_module           Enable my_module support])

if test "$PHP_MY_MODULE" != "no"; then
  dnl Action..
  PHP_EXTENSION(my_module, $ext_shared)
fi
```

Listing 9.2: Die Standard-config.m4

Wenn Sie mit den M4-Dateien noch nicht vertraut sind (jetzt ist sicher eine gute Zeit, sich mit ihnen vertraut zu machen), kann dies zunächst etwas verwirrend erscheinen, es ist aber tatsächlich sehr einfach.

> **Hinweis:** Alles, was das Präfix `dnl` hat, wird als Kommentar behandelt und nicht analysiert.

Die Datei `config.m4` ist zuständig für die Analyse der Befehlszeilenoption, die während der Konfiguration an `configure` übergeben werden. Dies bedeutet, dass sie überprüfen muss, ob erforderliche externe Dateien vorhanden sind, und ähnliche Konfigurations- und Installationsaufgaben übernimmt.

Die Standarddatei generiert zwei Konfigurationsbefehle im Skript `configure`: `--with-my_module` und `--enable-my_module`. Verwenden Sie die erste Option, wenn Sie auf externe Dateien referenzieren (so etwa die Anweisung `--with-apache`, um auf das Apache-Verzeichnis zu referenzieren). Verwenden Sie die zweite Option, wenn der Benutzer einfach nur entscheiden soll, ob Ihre Erweiterung freigegeben wird. Ganz egal welche Option Sie verwenden, Sie sollten die andere, nicht benötigte auskommentieren. Wenn Sie z.B. `--enable-my_module` verwenden, sollten Sie die Unterstützung für `--with-my_module` ausschalten und umgekehrt.

Standardmäßig akzeptiert die von `ext_skel` erzeugte Datei `config.m4` beide Anweisungen und gibt automatisch Ihre Erweiterung frei. Die Freigabe der Erweiterung geschieht über das Makro `PHP_EXTENSION`. Um das Standardverhalten zur Einbindung Ihres Moduls in die PHP-Binärdatei zu ändern – wenn der Benutzer dies wünscht (indem er explizit `--enable-my_module` oder `--with-my_module` angibt) – ändern Sie den Test für `$PHP_MY_MODULE` auf `== "yes"`:

```
if test "$PHP_MY_MODULE" == "yes"; then
dnl Action..
PHP_EXTENSION(my_module, $ext_shared)
fi
```

Damit müssten Sie `--enable-my_module` jedes Mal verwenden, wenn Sie PHP neu konfigurieren und kompilieren.

> **Hinweis:** Stellen Sie sicher, dass Sie `buildconf` jedes Mal ausführen, wenn Sie `config.m4` ändern!

Auf die M4-Makros, die für Ihre Konfigurationsskripte verfügbar sind, gehen wir später noch näher ein. Für den Moment verwenden wir einfach die Standarddateien. Alle Beispiel-Quelldateien auf der CD-ROM haben funktionierende `config.m4`-Dateien. Um sie in den Erstellungsprozess von PHP zu integrieren, brauchen Sie lediglich die Quellverzeichnisse in Ihr PHP-Verzeichnis `ext` kopieren, `buildconf` ausführen und dann die Beispielmodule mithilfe der entsprechenden `--enable-*`-Anweisungen mit `configure` einzubinden.

9.6 Erweiterungen erzeugen

Wir beginnen zunächst mit einer einfachen Erweiterung, die nichts weiter tut, als eine Funktion zu installieren, welche die Ganze Zahl, die es erhält, als Parameter auszugeben. Listing 9.3 zeigt den Quellcode.

```
/* include standard header */
#include "php.h"

/* declaration of functions to be exported */
ZEND_FUNCTION(first_module);

/* compiled function list so Zend knows what's in this module */
zend_function_entry firstmod_functions[] =
{
    ZEND_FE(first_module, NULL)
    {NULL, NULL, NULL}
};

/* compiled module information */
zend_module_entry firstmod_module_entry =
{
    "First Module",
    firstmod_functions,
    NULL, NULL, NULL, NULL, NULL,
    STANDARD_MODULE_PROPERTIES
};

/* implement standard "stub" routine to introduce ourselves to Zend */
```

```
#if COMPILE_DL
DLEXPORT zend_module_entry *get_module(void) { return(&firstmod_module_
➥entry); }
#endif

/* implement function that is meant to be made available to PHP */
ZEND_FUNCTION(first_module)
{
    zval **parameter;

    if((ZEND_NUM_ARGS() != 1) || (zend_get_parameters_ex(1, &parameter)
    ➥!= SUCCESS))
    {
        WRONG_PARAM_COUNT;
    }

    convert_to_long_ex(parameter);

    RETURN_LONG((*parameter)->value.lval);
}
```

Listing 9.3: Eine einfache Erweiterung

Dieser Code enthält ein vollständiges PHP-Modul. Wir werden auf den Quellcode in Kürze ausführlich eingehen, zunächst möchten wir aber den Erstellungsprozess erläutern. (Dies gibt den Ungeduldigen die Gelegenheit, zu experimentieren, bevor wir in die API-Diskussion einsteigen.)

9.7 Kompilierung von Modulen

Module können prinzipiell auf drei Arten kompiliert werden:

- über den bereitgestellten »make«-Mechanismus im Verzeichnis `dl`,
- über den bereitgestellten »make«-Mechanismus im Verzeichnis `ext`, über das auch dynamisch ladbare Module erstellt werden können oder
- durch manuelle Kompilierung der Quellen.

Die zweite Methode sollte definitiv favorisiert werden, da PHP 4.0 sie in einem anspruchsvollen Erstellungsprozess standardisiert hat. Die Tatsache, dass er so anspruchsvoll ist, ist auch sein Nachteil. Er ist zunächst etwas schwierig zu verstehen. Eine ausführlichere Einführung erhalten Sie später in diesem Kapitel, zunächst arbeiten wir mit den Standarddateien.

Der im Verzeichnis dl enthaltene make-Prozess hat etwas von Hackercode, ist veraltet und soll aus dem Quellverzeichnis entfernt werden. Zugegeben, es ist zunächst sehr viel einfacher, es zu verwenden, um dynamische Erweiterungen zu erzeugen, aber wir raten trotzdem davon ab, das Verzeichnis dl zu verwenden, da es nicht die Möglichkeiten des Verzeichnis ext hat und in Kürze sowieso gelöscht werden soll.

Die dritte Methode eignet sich für jene, denen, aus welchen Gründen auch immer, nicht das gesamte PHP-Quellverzeichnis zur Verfügung steht, die nicht Zugriff auf alle Dateien haben oder einfach nur auf ihrer Tastatur herumspielen wollen. Diese Fälle sollten zwar extrem selten sein, aber der Vollständigkeit halber gehen wir auch auf diese Methode ein.

9.7.1 Kompilierung mithilfe von Make

Zur Kompilierung der Beispielquellen über den Standardmechanismus kopieren Sie alle zugehörigen Unterverzeichnisse in das Verzeichnis ext Ihrer PHP-Quellstruktur. Anschließend führen Sie buildconf aus; dies erzeugt ein neues configure-Skript, das die entsprechenden Optionen enthält. Standardmäßig sind alle Beispiel-Quelldateien sind von der Kompilierung ausgeschlossen, Sie brauchen also keine Angst zu haben, dass Sie Ihren Erstellungsprozess zerstören.

Nachdem Sie buildconf ausgeführt haben, listet configure --help die folgenden zusätzlichen Module auf:

--enable-array_experiments	Aktiviert Array-Experimente.
--enable-call_userland	Aktiviert das Userland-Modul.
--enable-cross_conversion	Aktiviert das Cross Conversion-Modul.
--enable-firstmodule	Aktiviert das erste Modul.
--enable-infoprint	Aktiviert das Infoprint-Modul.
--enable-reference_test	Aktiviert das Referenztest-Modul.
--enable-resource_test	Aktiviert das Ressourcentest-Modul.
--enable-variable_creation	Aktiviert das Variablenerzeugungsmodul.

Die in Listing 9.3 aufgeführten Module können Sie aktivieren durch --enable-first_module oder --enable-first_module=yes.

9.7.2 Manuelles Kompilieren

Um Ihre Module manuell zu kompilieren, benötigen Sie die folgenden Befehle:

Aktion	Befehl
Kompilieren	`cc -fpic -DCOMPILE_DL=1 -I/usr/local/include -I. -I.. -I../Zend -c -o <Ihre_Objektdatei> <Ihre_C_Datei>`
Verknüpfen	`cc -shared -L/usr/local/lib -rdynamic -o <Ihre_Moduldatei> <Ihre_Objektdatei(en)>`

Der Befehl zur Kompilierung des Moduls weist den Compiler einfach an, einen positionsunabhängigen Code zu generieren (`-fpic` sollte nicht weggelassen werden), zusätzlich definiert er die Konstante `COMPILE_DL`, die dem Modulcode mitteilt, dass es als dynamisch ladbares Modul kompiliert wird (das Testmodul oben überprüft dies, wir gehen in Kürze darauf ein). Nach diesen Optionen, gibt er eine Reihe von Standard-Include-Pfaden an, die als Minimalausstattung zur Kompilierung der Quelldateien verwendet werden sollen.

Hinweis: Alle Include-Pfade im Beispiel sind relativ zum Verzeichnis `ext` angegeben. Wenn Sie die Kompilierung aus einem anderen Verzeichnis heraus durchführen, ändern Sie die Pfadnamen entsprechend. Als Komponenten benötigen Sie das PHP-Verzeichnis, das `Zend`-Verzeichnis und (sofern erforderlich) das Verzeichnis, in dem sich Ihr Modul befindet.

Der Verknüpfungsbefehl ist ebenfalls ein grundlegender Befehl, der eine Verknüpfung als dynamisches Modul vorschreibt.

Sie können Optimierungsoptionen in den Kompilierungsbefehl einschließen, auch wenn sie in diesem Beispiel weggelassen wurden (einige sind in der `makefile`-Schablone angegeben, die wir in einem früheren Abschnitt beschrieben haben).

Hinweis: Die manuelle Kompilierung und Verknüpfung als statisches Modul in die PHP-Binärdatei führt zu sehr langen Anweisungen, und wird daher hier nicht erörtert. (Es ist nicht sehr effizient, all diese Befehle einzutippen.)

9.8 Erweiterungen verwenden

Je nachdem, welchen Erstellungsprozess Sie ausgewählt haben, sollten Sie am Ende entweder eine PHP-Binärdatei, die mit Ihrem Webserver verknüpft ist (oder als CGI ausgeführt wird) oder eine `.so` (Shared Object)-Datei haben. Wenn Sie die Beispieldatei `first_module.c` als Shared Object kompiliert haben, sollte die erzeugte Datei `first_module.so` lauten. Um diese zu verwenden, müssen Sie sie zunächst an eine Stelle kopieren, an der PHP auf sie zugreifen kann. Zu Testzwecken können Sie die Datei in das Verzeichnis `htdocs` kopieren und sie mit der Quelle aus Listing 9.4 ausprobieren. Wenn Sie die Datei in die PHP-Binärdatei kompiliert haben, lassen Sie den `dl()`-Aufruf weg, denn die Funktionen des Moduls sind für Ihre Skripte sofort verfügbar.

Warnung: Zur Sicherheit sollten Sie Ihre dynamischen Module nicht in öffentlich zugänglichen Verzeichnisse speichern. Dies ist zwar möglich und vereinfacht das Testen, aber in Produktionsumgebungen sollten Sie diese in einem separaten Verzeichnis ablegen.

```
<?php

//dl("first_module.so");

$param = 2;
$return = first_module($param);

print("We sent \"$param\" and got \"$return\"");

?>
```

Listing 9.4: Eine Testdatei für first_module.so

Wenn Sie diese PHP-Datei in Ihrem Webbrowser aufrufen, erhalten Sie die in Abbildung 9.3 gezeigte Ausgabe.

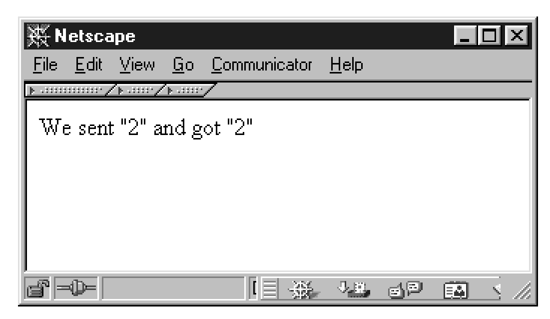

Abbildung 9.3: Ausgabe von first_module.php

Sofern erforderlich wird das dynamisch ladbare Modul durch Aufruf der Funktion `dl()` geladen. Diese Funktion sucht nach dem angegebenen Shared Object, lädt es und stellt PHP seine Funktionen zur Verfügung. Das Modul exportiert die Funktion `first_module()`, die einen einzigen Parameter besitzt, konvertiert diesen in eine ganze Zahl und gibt das Ergebnis der Konvertierung aus.

Wenn Sie so weit gekommen sind, herzlichen Glückwunsch! Sie haben soeben Ihre erste Erweiterung von PHP generiert.

9.9 Fehlersuche

Genau genommen ist bei der Kompilierung von statischen oder dynamischen Modulen kaum eine Fehlersuche möglich. Das einzige Problem, das auftreten könnte, ist, dass sich der Compiler sich über fehlende Definitionen oder etwas Ähnliches beschwert. Stellen Sie in diesem Fall sicher, dass alle Headerdateien verfügbar sind und dass Sie Ihren Pfad im Kompilierbefehl korrekt angegeben haben. Um sicher zu gehen, dass alles an der richtigen Stelle ist, extrahieren Sie ein sauberes PHP-Quellverzeichnis und nutzen Sie die automatische Erstellung im Verzeichnis `ext` mit den frischen Dateien von der CD-ROM. Dies garantiert eine sichere Kompilierungsumgebung. Wenn dieses Verfahren fehlschlägt, versuchen Sie es mit der manuellen Kompilierung.

Möglicherweise beschwert sich PHP auch über fehlende Funktionen in Ihrem Modul. (Dies solle bei den Beispielquelldateien nicht passieren, wenn Sie diese nicht verändert haben.) Wenn die Namen der externen Funktionen, auf die Sie mit Ihrem Modul zugreifen möchten, falsch geschrieben sind, verbleiben Sie als »nicht verknüpfte Symbole« in der Symboltabelle. Während des dynamischen Ladens und Verknüpfens durch PHP werden sie aufgrund der Schreibfehler nicht aufgelöst. Es gibt keine entsprechenden Symbole in der Hauptbinärdatei. Suchen Sie nach nicht korrekten Deklarationen in Ihrer Moduldatei oder falsch geschriebenen externen Referenzen. Beachten Sie, dass dieses Problem nur für dynamisch ladbare Module gilt. Bei statischen Modulen tritt es nicht auf. Fehler in statischen Modulen zeigen sich während der Kompilierung.

9.10 Einblick in die Quellen

Nachdem Sie eine sichere Erstellungsumgebung haben und in der Lage sind, die Module in die PHP-Dateien einzubinden, sollten wir uns ansehen, wie alles zusammen funktioniert.

9.10.1 Modulstruktur

Alle PHP-Module haben denselben Aufbau:

- eingebundene Headerdatei (zur Integration aller notwendigen Makros, API-Definitionen usw.)
- C-Deklaration der exportierten Funktionen (zur Deklaration des Zend-Funktionsblock)
- Deklaration des Zend-Funktionsblock
- Deklaration des Zend-Modulblocks
- Einbindung von `get_module()`
- Einbindung aller exportierten Funktionen

9.10.2 Einbindung der Headerdatei

Die einzige Headerdatei, die Sie wirklich für Ihre Module einbinden müssen, ist `php.h`, die im PHP-Verzeichnis abgelegt ist. Diese Datei macht alle Makros und API-Definitionen, die Sie zum Generieren neuer Module benötigen, für Ihren Code verfügbar.

> **Tipp:** Es ist empfehlenswert, eine separate Headerdatei für Ihr Modul zu erstellen, welche die modulspezifischen Funktionen enthält. Die Headerdatei sollte alle Deklarationen für exportierte Funktionen sowie `php.h` enthalten.

9.10.3 Deklaration exportierter Funktionen

Um Funktionen zu deklarieren, die exportiert werden sollen (d.h. für PHP als neue direkte Funktionen verfügbar machen), bietet Zend eine Reihe von Makros. Eine Deklaration sieht z.B. folgendermaßen aus:

```
ZEND_FUNCTION(my_function);
```

`ZEND_FUNCTION` deklariert eine neue C-Funktion, die mit der internen API von Zend übereinstimmt. Das bedeutet, dass die Funktion den Typ `void` hat und als Parameter `INTERNAL_FUNCTION_PARAMETERS` (ein weiteres Makro) akzeptiert. Zusätzlich wird dem Funktionsnamen das Präfix `zend_if` vorangestellt. Die sofort expandierte Version der oben definierten Funktion sieht folgendermaßen aus:

```
void zend_if_my_function(INTERNAL_FUNCTION_PARAMETERS);
```

Die Erweiterung von `INTERNAL_FUNCTION_PARAMETERS` führt zu folgendem Ergebnis:

```
void zend_if_my_function(int ht, zval *return_value,
                         ➥zval *this_ptr, int return_value_used,
                         ➥zend_executor_globals *executor_globals);
```

Da der Interpreter und Executor-Kern von dem PHP-Hauptpaket abgetrennt wurde, hat sich eine zweite API entwickelt, welche Makros und Funktionssätze definiert: die Zend-API. Da die Zend-API inzwischen einige der Aufgaben übernommen hat, die zuvor zu PHP gehörten, wurden viele PHP-Funktionen auf Makros reduziert, die aus einfachen Aufrufen von Funktionen der Zend-API bestehen. Die empfohlenen Praxis ist, die Zend-API wo immer möglich zu verwenden, da die alte API nur aus Kompatibilitätsgründen erhalten bleibt. So sind z. B: die Typen `zval` und `pval` identisch. `zval` ist die Zend-Definition, während `pval` die PHP-Definition ist (genau genommen ist `pval` ein Alias für `zval`). Da das Makro `INTERNAL_FUNCTION_PARAMETERS` ein Zend-Makro ist, enthält die obige Deklaration `zval`. Wenn Sie Programmcode schreiben, sollten Sie stets `zval` verwenden, um der neuen Zend-API zu entsprechen.

Die Parameterliste dieser Deklaration ist sehr wichtig. Sie sollten sich diese Parameter merken (siehe Tabelle 9.1 für die Beschreibungen).

Parameter	Beschreibung
`ht`	Anzahl der Argumente, die an die Zend-Funktion übergeben werden. Sie sollten ihn nicht direkt benutzen, sondern statt dessen `ZEND_NUM_ARGS()` verwenden, um den Wert zu erhalten.
`return_value`	Diese Variable wird verwendet, um die von der Funktion zurückgegebene Werte wieder an PHP zurück zu übergeben. Den Zugriff auf diese Variable erreichen Sie am besten über die vordefinierten Makros. Eine Beschreibung der vordefinierten Makros finden Sie weiter unten.
`this_ptr`	Mit dieser Variable können Sie auf das Objekt zugreifen, in dem die Funktion enthalten ist, sofern sie in einem Objekt verwendet wird. Verwenden Sie die Funktion `getThis()` um diesen Zeiger zu erhalten.
`return_value_used`	Dieser Flag gibt an, ob ein möglicher Ausgabewert aus dieser Funktion tatsächlich vom anrufenden Skript verwendet wird. `0` gibt an, dass der Ausgabewert nicht verwendet wird, `1` gibt an, dass der Anrufer einen Rückgabewert erwartet. Sie können diesen Flag auswerten, um zu überprüfen, ob die Funktion korrekt verwendet wurde oder um die Geschwindigkeit zu optimieren, wenn die Rückgabe eines Wertes aufwendige Operationen erfordern würden (als Beispiel sehen Sie sich an, wie `array.c` ihn verwendet).
`executor_globals`	Diese Variable zeigt auf die globale Einstellung der Zend-Engine. Sie ist beispielsweise zur Erzeugung neuer Variablen nützlich (mehr dazu später). Die globalen Variablen des Executors können über das Makro `ELS_FETCH()` in Ihre Funktionen integriert werden.

Tab. 9.1: Zend-Parameter für Funktionen, die mit PHP aufgerufen werden

9.10.4 Deklarationen des Zend-Funktionsblocks

Nachdem Sie die zu exportierenden Funktionen deklariert haben, müssen Sie sie auch für Zend einführen. Die Einführung der Funktionsliste geschieht über einen Array von `zend_function_entry`. Dieser Array enthält alle Funktionen, die bereitgestellt werden sollen, wobei der Funktionsname so angegeben ist, wie er in PHP erscheinen soll, und der Name, wie er in der C-Quelle definiert ist. Intern wird `zend_function_entry` wie in Listing 9.5 gezeigt definiert.

```
typedef struct _zend_function_entry {
    char *fname;
    void (*handler)(INTERNAL_FUNCTION_PARAMETERS);
    unsigned char *func_arg_types;
} zend_function_entry;
```

Listing 9.5: Interne Deklaration von `zend_function_entry`

Die folgende Tabelle beschreibt die Einträge:

Eintrag	Beschreibung
`fname`	Gibt den Funktionsnamen an, wie er in PHP gesehen wird (z.B. `fopen`, `mysql_connect` oder in unserem Beispiel `first_module`).
`handler`	Zeiger auf die C-Funktion, die für die Verwaltung von Aufrufen dieser Funktion zuständig ist, siehe z.B. das zuvor besprochene Standardmakro `INTERNAL_FUNCTION_PARAMETERS`.
`func_arg_types`	Ermöglicht Ihnen, bestimmte Parameter zu markieren, so dass sie per Referenz übergeben werden müssen. In der Regel sollten Sie diesen Eintrag auf `NULL` setzen.

Im obigen Beispiel sieht die Deklaration folgendermaßen aus:

```
zend_function_entry firstmod_functions[] =
{
    ZEND_FE(first_module, NULL)
    {NULL, NULL, NULL}
};
```

Wie Sie sehen, muss der letzte Eintrag in dieser Liste stets `{NULL, NULL, NULL}` sein. Dieser Marker muss gesetzt werden, damit Zend weiß, wann das Ende der Liste der exportierten Funktionen erreicht ist.

> **Hinweis:** Die vordefinierten Makros können Sie nicht für die Endmarkierung verwenden, da diese versuchen würden, eine Funktion namens »NULL« zu referenzieren!

Das Makro ZEND_FE wird einfach zu einen Struktureintrag in zend_function_entry expandiert. Beachten Sie, dass diese Makros ein spezielles Namensschema für Ihre Funktionen einführen. Ihre C-Funktionen erhalten das Präfix zend_if_. Demnach referenziert ZEND_FE(first_module) auf eine C-Funktion namens zend_if_first_module(). Behalten Sie dies im Hinterkopf, wenn Sie Makros und manuell kodierte Einträge mischen möchten (was keine gute Praxis ist).

Tipp: Kompilierungsfehler, die auf Funktionen namens zend_if_*() verweisen, beziehen sich auf Funktionen, die mit ZEND_FE definiert wurden.

Tabelle 9.2 zeigt eine Liste aller Makros, die Sie zur Definition von Funktionen verwenden können.

Makroname	Beschreibung
ZEND_FE(Name, Arg_typen)	Definiert einen Funktionseintrag des Namens *Name* in zend_function_entry. Benötigt eine entsprechende C-Funktion. *Arg_typen* muss auf NULL gesetzt sein. Diese Funktion verwendet eine automatische Generierung von C-Funktionsnamen, durch Voranstellung des Präfix zend_if_ vor den PHP-Funktionsnamen. ZEND_FE("first_module", NULL) führt beispielsweise die Funktion first_module() in PHP ein, und verknüpft sie mit der C-Funktion zend_if_first_module(). Verwenden Sie dieses Makro in Verbindung mit ZEND_FUNCTION.
ZEND_NAMED_FE (*PHP_Name*, *Name*, *Arg_typen*)	Definiert eine Funktion, die über den Namen *PHP_Name* für PHP verfügbar ist, und verknüpft diese mit der entsprechenden C-Funktion *Name*. *Arg_typen* muss auf NULL gesetzt sein. Verwenden Sie diese Funktion, wenn Sie das von ZEND_FE eingeführte Namenspräfix nicht verwenden möchten. Verwenden Sie dieses Makro in Verbindung mit ZEND_NAMED_FUNCTION.
ZEND_FALIAS (Name, Alias, Arg_typen)	Definiert einen Alias namens *Alias* für *Name*. *Arg_typen* muss auf NULL gesetzt sein. Benötigt keine entsprechende C-Funktion, referenziert statt dessen auf das Aliasziel.
PHP_FE(Name, Arg_typen)	Alte PHP-API, äquivalent zu ZEND_FE
PHP_NAMED_FE (Laufzeit_name, Name, Arg_typen)	Alte PHP-API, äquivalent zu ZEND_NAMED_FE

Tab. 9.2: Makros zur Definition von Funktionen

Hinweis: ZEND_FE kann nicht zusammen mit PHP_FUNCTION verwendet werden, genauso wenig wie PHP_FE mit ZEND_FUNCTION. Zulässig ist jedoch, ZEND_FE und ZEND_FUNCTION mit PHP_FE und PHP_FUNCTION zu kombinieren, wenn Sie bei den zu deklarierenden Funktionen stets im selben Makrosatz bleiben. Die Vermischung ist jedoch nicht empfehlenswert. Statt dessen sollten Sie nur ZEND_*-Makros verwenden.

9.10.5 Deklaration des Zend-Modulblocks

Dieser Block wird in der Struktur zend_module_entry gespeichert und enthält alle notwendigen Informationen, um den Inhalt des Moduls für Zend zu beschreiben. Die internen Definition dieses Moduls zeigt Listing 9.6.

```
typedef struct _zend_module_entry zend_module_entry;

struct _zend_module_entry {
    char *name;
    zend_function_entry *functions;
    int (*module_startup_func)(INIT_FUNC_ARGS);
    int (*module_shutdown_func)(SHUTDOWN_FUNC_ARGS);
    int (*request_startup_func)(INIT_FUNC_ARGS);
    int (*request_shutdown_func)(SHUTDOWN_FUNC_ARGS);
    void (*info_func)(ZEND_MODULE_INFO_FUNC_ARGS);
    int (*global_startup_func)(void);
    int (*global_shutdown_func)(void);

[ Rest of the structure is not interesting here ]

};
```

Listing 9.6: Interne Deklaration von zend_module_entry.

Die folgende Tabelle beschreibt die Einträge.

Eintrag	Beschreibung
name	Enthält den Namen des Moduls (z.B. "File functions", "Socket functions", "Crypt" usw.). Dieser Name erscheint in phpinfo() im Abschnitt »Additional Modules.«
functions	Zeigt auf den Zend-Funktionsblock, der im vorherigen Abschnitt besprochen wurde.

Eintrag	Beschreibung
module_startup_func	Diese Funktion wird bei der Initialisierung des Moduls einmal aufgerufen und kann für einmalige Initialisierungsschritte verwendet werden (z. B. die anfängliche Speicherzuweisung usw.). Bei einem Fehler während der Initialisierung soll die Funktion FAILURE zurückgegeben werden, ansonsten SUCCESS. Um dieses Feld als nicht benutzt zu markieren, verwenden Sie NULL. Zur Deklaration einer Funktion verwenden Sie das Makro ZEND_MINIT.
module_shutdown_func	Diese Funktion wird bei der Modul-Deinitialisierung einmal aufgerufen, und kann für einmalige Deinitialisierungschritte verwendet werden (wie etwa die Aufhebung einer Speicherzuweisung). Sie bildet das Gegenstück zu module_startup_func(). Bei einem Fehler während der Deinitialisierung soll FAILURE zurückgegeben werden, ansonsten SUCCESS. Um dieses Feld als nicht benutzt zu markieren, verwenden Sie NULL. Zur Deklaration einer Funktion verwenden Sie das Makro ZEND_MSHUTDOWN.
request_startup_func	Diese Funktion wird einmal bei jeder Seitenanfrage aufgerufen, und kann für einmalige Initialisierungsschritte verwendet werden, die zur Verarbeitung einer Anfrage erforderlich sind. Ein Fehlschlagen zeigen Sie durch Ausgabe von FAILURE an, ansonsten soll SUCCESS zurückgegeben werden.
request_shutdown_func	Diese Funktion wird einmal nach jeder Seitenanfrage aufgerufen. Sie stellt das Gegenstück zu request_startup_func() dar. Um ein Fehlschlagen anzuzeigen, geben Sie FAILURE zurück, ansonsten SUCCESS. Hinweis: Da dynamisch ladbare Module nur bei Seitenanfragen geladen werden, folgt auf die Anfragebeendigungsfunktion direkt ein Aufruf der Modulbeendigungsroutine (beides wird gleichzeitig deinitialisiert). Um dieses Feld als nicht benutzt zu markieren, verwenden Sie NULL. Zur Deklaration einer Funktion verwenden Sie das Makro ZEND_RSHUTDOWN.

Eintrag	Beschreibung
info_func	Wenn phpinfo() in einen Skript aufgerufen wird, durchläuft Zend alle geladenen Module und ruft diese Funktion auf. Jedes Modul hat dann die Gelegenheit, seinen eigenen »Fingerabdruck« auf der Ausgabeseite zu hinterlassen. Im Allgemeinen wird diese Funktion verwendet, um Umgebungs- und statistische Daten zu hinterlassen. Um dieses Feld als nicht benutzt zu markieren, verwenden Sie NULL. Zur Deklaration einer Funktion verwenden Sie das Makro ZEND_MINFO.
global_startup_func	Die globale Initialisierungsfunktionen werden nur selten verwendet. In der Regel sollten Sie die restliche Struktur mit dem Makro STANDARD_MODULE_PROPERTIES durchsuchen. Um dieses Feld als nicht benutzt zu markieren, verwenden Sie NULL. Zur Deklaration einer Funktion verwenden Sie das Makro ZEND_GINIT.
global_shutdown_func	Um dieses Feld als nicht benutzt zu markieren, verwenden Sie NULL. Zur Deklaration einer Funktion verwenden Sie das Makro ZEND_GSHUTDOWN.
Sonstige Strukturelemente	Diese werden intern verwendet und können über das Makro STANDARD_MODULE_PROPERTIES_EX im voraus ausgeführt werden. Sie sollten ihnen keine Werte zuweisen. Verwenden Sie STANDARD_MODULE_PROPERTIES_EX nur, wenn Sie globale Initialisierungs- und Deinitialisierungsfunktionen verwenden. Ansonsten verwenden Sie direkt STANDARD_MODULE_PROPERTIES.

Hinweis: Da dynamisch ladbare Module nur bei Seitenanfragen geladen werden, wird die Anfrageinitialisierungsfunktion direkt nach der Modulstartfunktion aufgerufen (beide Initialisierungen werden zur selben Zeit durchgeführt). Um dieses Feld als nicht benutzt zu markieren, verwenden Sie NULL. Zur Deklaration einer Funktion verwenden Sie das Makro ZEND_RINIT.

In unserem Beispiel sieht der Modulblock folgendermaßen aus:

```
zend_module_entry firstmod_module_entry =
{
    "First Module",
    firstmod_functions,
    NULL, NULL, NULL, NULL, NULL,
    STANDARD_MODULE_PROPERTIES
};
```

Dies ist im Prinzip die einfachste und minimalste Anzahl an Werten, die Sie einsetzen können. Der Name des Moduls wird auf `First Module` gesetzt. Anschließend wird die Funktionsliste referenziert und danach alle Initialisierungs- und Deinitialisierungsfunktionen als nicht benutzt markiert.

Zu Referenzzwecken finden Sie in Tabelle 9.3 eine Liste der Makros, die in den deklarierten Initialisierungs- und Deinitialisierungsfunktionen vorkommen. In unserem einfachen Beispiel wurden sie bisher noch nicht eingesetzt, sie werden aber später noch eine Rolle spielen. Sie sollten diese Makros verwenden, um Ihre Initialisierungs- und Deinitialisierungsfunktionen zu deklarieren, da hierfür spezielle Argumente übergeben werden müssen (`INIT_FUNC_ARGS` und `SHUTDOWN_FUNC_ARGS`), die bei Verwendung der vordefinierten Makros automatisch in der Funktionsdeklaration enthalten sind. Wenn Sie Ihre Funktionen manuell deklarieren und der PHP-Entwickler beschließt, dass eine Änderung in der Argumentenliste notwendig ist, müssen Sie Ihre Modulquellen ebenfalls ändern, um kompatibel zu bleiben.

Makro	Beschreibung
`ZEND_MINIT(module)`	Deklariert eine Funktion zur Modulinitialisierung. Der erzeugte Name lautet `zend_minit_<module>` (z. B. `zend_minit_first_module`). Zu verwenden in Verbindung mit `ZEND_MINIT_FUNCTION`.
`ZEND_MSHUTDOWN(module)`	Deklariert eine Funktion zur Moduldeinitialisierung. Der erzeugte Name lautet `zend_mshutdown_<module>` (z. B. `zend_mshutdown_first_module`). Zu verwenden in Verbindung mit `ZEND_MSHUTDOWN_FUNCTION`.
`ZEND_RINIT(module)`	Deklariert eine Funktion zur Anfrageinitialisierung. Der erzeugte Name lautet `zend_rinit_<module>` (z. B. `zend_rinit_first_module`). Zu verwenden in Verbindung mit `ZEND_RINIT_FUNCTION`.
`ZEND_RSHUTDOWN(module)`	Deklariert eine Funktion zur Anfragedeinitialisierung. Der erzeugte Name lautet `zend_rshutdown_<module>` (z. B. `zend_rshutdown_first_module`). Zu verwenden in Verbindung mit `ZEND_RSHUTDOWN_FUNCTION`.
`ZEND_GINIT(module)`	Deklariert eine Funktion zur globalen Initialisierung. Der erzeugte Name lautet `zend_ginit_<module>` (z. B. `zend_ginit_first_module`). Zu verwenden in Verbindung mit `ZEND_GINIT_FUNCTION`.

Tab. 9.3: Makros zur Deklaration der Initialisierungs- und Deinitialisierungsfunktionen

Makro	Beschreibung
ZEND_GSHUTDOWN(module)	Deklariert eine Funktion zur globalen Deinitialisierung. Der erzeugte Name lautet zend_gshutdown_<module> (z.B. zend_gshutdown_first_module). Zu verwenden in Verbindung mit ZEND_GSHUTDOWN_FUNCTION.
ZEND_MINFO(module)	Deklariert eine Funktion zur Ausgabe von Modulinformation, die beim Aufruf von phpinfo() verwendet wird. Der erzeugte Name lautet zend_info_<module> (z.B. zend_info_first_module). Zu verwenden in Verbindung mit ZEND_MINFO_FUNCTION.

Tab. 9.3: Makros zur Deklaration der Initialisierungs- und Deinitialisierungsfunktionen

9.10.6 Einbettung von get_module()

Diese Funktion ist speziell für alle dynamisch ladbaren Module gedacht. Betrachten wir zunächst, wie sie installiert wird:

```
#if COMPILE_DL
DLEXPORT zend_module_entry *get_module(void) { return(&firstmod_module
➥_entry); }
#endif
```

Die Funktionsdefinition ist in eine bedingte Kompilierungsanweisung eingefasst. Dies ist erforderlich, da die Funktion get_module() nur dann benötigt wird, wenn Ihr Modul als dynamische Erweiterung generiert wird. Durch Angabe der Definition von COMPILE_DL im Compilerbefehl (eine Erörterung der zur Generierung einer dynamischen Erweiterung erforderlichen Kompilierungsbefehle finden Sie weiter oben) können Sie Ihr Modul anweisen, ob Sie es als dynamische Erweiterung oder als eingebautes Modul realisieren wollen. Bei einem eingebauten Modul wird get_module() einfach nicht eingebunden.

get_module() wird von Zend aufgerufen, wenn das Modul geladen wird. Zur Verdeutlichung stellen Sie sich einfach vor, die Funktion würde durch den dl()-Aufruf in Ihrem Skript gestartet. Sie dient dazu, den Modulinformationsblock wieder an Zend zu übergeben, um die Engine über den Modulinhalt zu informieren.

Wenn Sie die Funktion get_module() nicht in Ihrem dynamisch ladbaren Modul einbinden, gibt Zend eine Fehlermeldung aus, sobald sie versuchen darauf zuzugreifen.

9.10.7 Einbindung aller exportierten Funktionen

Schließlich werden alle exportierten Funktion implementiert. Für unsere Beispielfunktion aus `first_module` sieht das folgendermaßen aus:

```
ZEND_FUNCTION(firstmodule)
{
    zval **parameter;

    if((ZEND_NUM_ARGS() != 1) || (zend_get_parameters_ex(1, &parameter)
    ➥!= SUCCESS))
    {
        WRONG_PARAM_COUNT;
    }

    convert_to_long_ex(parameter);

    RETURN_LONG((*parameter)->value.lval);
}
```

Die Funktionsdeklaration geschieht über `ZEND_FUNCTION`, welche in der Funktionseintragstabelle (auf die wir bereits eingegangen sind) `ZEND_FE` entspricht.

Nach den Deklarationen folgt der Programmcode zur Überprüfung und zum Auslesen der Funktionsargumente, der Konvertierung von Argumenten und der Generierung von Rückgabewerten (mehr dazu später).

9.10.8 Zusammenfassung

Im Prinzip ist dies alles – mehr gibt es zur Implementierung von PHP-Modulen nicht zu sagen. Eingebaute Module sind ähnlich aufgebaut wie dynamische Module. Mit den hier präsentierten Informationen sollten Sie in der Lage sein, Probleme zu bewältigen, die in den Quelldateien der PHP-Module auftreten.

In den folgenden Abschnitten gehen wir darauf ein, wie Sie die PHP-Interna verwenden, um leistungsfähige Erweiterungen zu generieren.

9.11 Annahme von Argumenten

Einer der wichtigsten Aspekte bei Spracherweiterungen betrifft die Annahme und Handhabung von Daten, die durch Argumente übergeben werden. Die meisten Erweiterungen dienen zur Verarbeitung spezieller Eingabedaten (oder benötigen Parameter zur Durchführung ihrer spezifischen Aktionen). Funktionsargumente sind eigentlich die einzig wahre Möglichkeit, um Daten zwischen der PHP- und der C-Ebene auszutauschen. Natürlich gibt es auch

noch die Möglichkeit, Daten über vordefinierte globale Variablen auszutauschen (hierauf gehen wir später ein), dies sollten Sie jedoch unter allen Umständen vermeiden, da es von schlechter Programmierpraxis zeugt. Einzelheiten finden Sie in Kapitel 1 »Entwicklungskonzepte«.

PHP verwendet keine formalen Funktionsdeklarationen; daher ist die Aufrufsyntax stets vollständig dynamisch und wird nicht auf Fehler überprüft. Die Überprüfung der Aufrufsyntax bleibt dem Benutzer überlassen. Zum Beispiel kann eine Funktion einmal mit nur einem Argument und ein anderes mal mit vier Argumenten aufgerufen werden – beide Aufrufe sind syntaktisch absolut korrekt.

9.11.1 Anzahl der Argumente feststellen

Da PHP keine formalen Funktionsdefinitionen besitzt, die eine Unterstützung für die Überprüfung der Aufrufsyntax bieten, aber variable Argumente verwendet, müssen Sie manchmal ermitteln, mit wie vielen Argumenten Ihre Funktion aufgerufen wurde. Hierfür können Sie das Makro ZEND_NUM_ARGS verwenden. In früheren PHP-Versionen las dieses Makro die Anzahl an Argumenten aus, mit denen die Funktion aufgerufen wurde. Als Basis diente dafür der Eintrag ht in der Hash-Tabelle der Funktion, der in der Liste INTERNAL_FUNCTION_PARAMETERS übergeben wurde Da ht inzwischen die Anzahl der Argumente selbst angibt, die an die Funktion übergeben wurden, ist ZEND_NUM_ARGS zu einem Dummy-Makro mutiert (siehe seine Definition in zend_API.h). Es ist aber immer noch gute Praxis es zu verwenden, um zu zukünftigen Änderungen der Aufrufschnittstelle kompatibel zu bleiben.

Hinweis: Die alte PHP-Äquivalente dieses Makros heißt ARG_COUNT.

Der folgende Programmcode überprüft die richtige Anzahl der Argumente:

```
if(ZEND_NUM_ARGS() != 2)
    WRONG_PARAMETER_COUNT;
```

Wenn diese Funktion nicht mit zwei Argumenten aufgerufen wird, würde eine Fehlermeldung ausgegeben.. Der obige Programmabschnitt verwendet das Makro WRONG_PARAMETER_COUNT, das zur Generierung einer Standardfehlermeldung verwendet werden kann (siehe Abbildung 9.4).

Diese Makro gibt eine Standardfehlermeldung aus und kehrt anschließend zum Anrufer zurück. Seine Definition finden Sie in zend_API.h; sie sieht folgendermaßen aus:

```
ZEND_API void wrong_param_count(void);

#define WRONG_PARAM_COUNT { wrong_param_count(); return; }
```

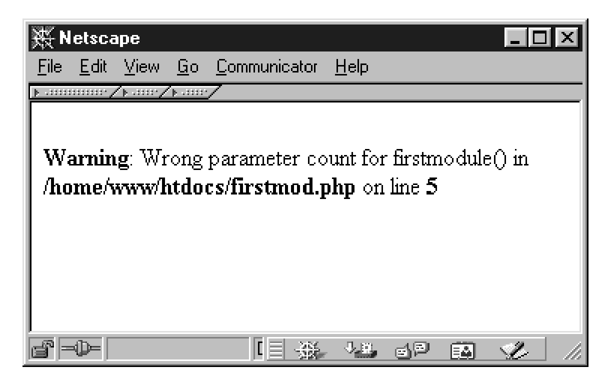

Abbildung 9.4: WRONG_PARAMETER_COUNT in Aktion.

Wie Sie sehen, wird in dieser Definition eine internen Funktion namens `wrong_param_count()` aufgerufen, die für die Ausgabe der Warnung zuständig ist. Auf Einzelheiten für die Erstellung benutzerspezifischer Fehlermeldungen gehen wir später im Abschnitt »Ausdrucken von Informationen« ein.

9.11.2 Auslesen von Argumenten

Nachdem Sie die Anzahl der Argumente überprüft haben, müssen Sie auf diese Argumente zugreifen können. Hierfür steht Ihnen `zend_get_parameters_ex()` zur Verfügung.

```
zval **parameter;

if(zend_get_parameters_ex(1, &parameter) != SUCCESS)
    WRONG_PARAMETER_COUNT;
```

Alle Argumente werden im Container `zval` gespeichert, auf den zweimal gezeigt werden muss. Das obige Beispiel zeigt den Versuch, ein Argument zu lesen und es über den Zeiger `parameter` bereitzustellen.

`zend_get_parameters_ex()` nimmt mindestens zwei Argumente an. Das erste Argument gibt die zu lesende Anzahl der Argumente an. Dies sollte der Anzahl von Argumenten entsprechen, mit welcher die Funktion aufgerufen wurde – deswegen sollten Sie die korrekte Aufrufsyntax überprüfen. Das zweite Argument (und alle weiteren Argumente) sind Zeiger auf Zeiger auf Zeiger auf `zval`s. (Verwirrend – nicht wahr?) Diese Zeiger sind alle erforderlich, da Zend intern mit `**zval` arbeitet. Um einen lokalen `**zval` in unserer Funktion auszurichten, benötigt `zend_get_parameters_ex()` einen Zeiger darauf.

Als Rückgabewert von zend_get_parameters_ex() wird entweder SUCCESS oder FAILURE ausgegeben, die eine erfolgreiche (bzw. fehlgeschlagene) Argumentenverarbeitung angeben. Ein Fehlschlagen geht wahrscheinlich auf eine falsche Angabe der Argumentenanzahl zurück. In diesem Falle sollten Sie die Funktion mit WRONG_PARAMETER_COUNT beenden.

Um mehr als ein Argument zu lesen, können sie ein ähnliche Konstruktion verwenden:

```
zval **param1, **param2, **param3, **param4;

if(zend_get_parameters_ex(4, &param1, &param2, &param3, &param4) !=
➥SUCCESS)
    WRONG_PARAMETER_COUNT;
```

zend_get_parameters_ex() überprüft nur, ob Sie versuchen, zu viele Parameter auszulesen. Wenn die Funktion mit fünf Argumenten aufgerufen wird, Sie aber nur drei von ihnen mit zend_get_parameters_ex() abfragen, erhalten Sie keine Fehlermeldung, sondern die ersten drei Parameter. Bei nachfolgenden Aufrufen von zend_get_parameters_ex() werden nicht die verbleibenden Argumente ausgelesen, sondern wieder dieselben Argumente.

9.11.3 Verarbeitung mit einer variablen Anzahl von Argumenten/optionalen Parametern

Wenn Ihre Funktion eine variable Anzahl von Argumenten annehmen soll, bieten die soeben beschriebenen Codeabschnitte manchmal nur suboptimale Lösungen. Für jede mögliche Anzahl von Argumenten müssten Sie eine eigene Zeile schreiben, die zend_get_parameters_ex() aufruft. Dies ist häufig nicht zufriedenstellend.

In diesem Falle können Sie die Funktion zend_get_parameters_array_ex() verwenden, welche die Anzahl der auszulesenden Parameter angibt, sowie ein Array, in dem Sie diese speichern.

```
zval **parameter_array[4];

/* get the number of arguments */
argument_count = ZEND_NUM_ARGS();

/* see if it satisfies our minimal request (2 arguments) */
/* and our maximal acceptance (4 arguments) */
if(argument_count < 2 || argument_count > 5)
    WRONG_PARAMETER_COUNT;

/* argument count is correct, now retrieve arguments */
```

```
if(zend_get_parameters_array_ex(argument_count, parameter_array) !=
➥SUCCESS)
    WRONG_PARAMETER_COUNT;
```

Zunächst wird die Anzahl der Argumente überprüft um sicherzustellen, dass sie innerhalb des zulässigen Bereichs liegt. Danach wird `zend_get_parameters_array_ex()` verwendet, um `parameter_array` mit gültigen Zeigern auf die Argumentwerte zu füllen.

Eine clevere Realisierung dieser Methode stellt der Code dar, der PHPs `fsockopen()` handhabt, die sich im `ext/standard/fsock.c` befindet. Siehe hierzu auch Listing 9.7. Seien Sie nicht beunruhigt, wenn Sie noch nicht alle Funktionen kennen, die in diesem Quellcode verwendet werden. Wir kommen in Kürze auf sie zurück.

```
pval **args[5];
int *sock=emalloc(sizeof(int));
int *sockp;
int arg_count=ARG_COUNT(ht);
int socketd = -1;
unsigned char udp = 0;
struct timeval timeout = { 60, 0 };
unsigned short portno;
unsigned long conv;
char *key = NULL;
FLS_FETCH();

if (arg_count > 5 || arg_count < 2 ||
➥zend_get_parameters_array_ex(arg_count,args)==FAILURE) {
    CLOSE_SOCK(1);
    WRONG_PARAM_COUNT;
}

switch(arg_count) {
    case 5:
        convert_to_double_ex(args[4]);
        conv = (unsigned long) ((*args[4])->value.dval * 1000000.0);
        timeout.tv_sec = conv / 1000000;
        timeout.tv_usec = conv % 1000000;
        /* fall-through */
    case 4:
        if(!ParameterPassedByReference(ht,4)) {
            php_error(E_WARNING,"error string argument to fsockopen not
            ➥passed by reference");
        }
        pval_copy_constructor(*args[3]);
```

```
        (*args[3])->value.str.val = empty_string;
        (*args[3])->value.str.len = 0;
        (*args[3])->type = IS_STRING;
        /* fall-through */
    case 3:
        if(!ParameterPassedByReference(ht,3)) {
            php_error(E_WARNING,"error argument to fsockopen not passed by
            ➥reference");
        }
        (*args[2])->type = IS_LONG;
        (*args[2])->value.lval = 0;
        break;
}

convert_to_string_ex(args[0]);
convert_to_long_ex(args[1]);
portno = (unsigned short) (*args[1])->value.lval;

key = emalloc((*args[0])->value.str.len + 10);
```

Listing 9.7: Eine PHP-Realisierung variabler Argumente mit fsockopen( )

`fsockopen()` nimmt zwei, drei, vier oder fünf Parameter an. Nach den obligatorischen Variablendeklarationen überprüft die Funktion den Bereich der Argumente auf Gültigkeit. Anschließend verwendet sie ein Fall-Through-Mechanismus mit der Anweisung `switch()`, um alle Argumente zu verarbeiten. Die Anweisung `switch()` beginnt mit der größten Anzahl von übergebenen Argumenten (fünf). Danach verarbeitet sie automatisch die Fälle mit vier übergebenen Argumenten, dann jene mit drei, wobei in allen Stufen das ansonsten obligatorische Schlüsselwort `break` weggelassen wird. Nachdem der letzte Fall verarbeitet wurde, beendet sie die Anweisung `switch()` und führt die Minimalverarbeitung der Argumente durch, wenn die Funktion mit nur zwei Argumenten aufgerufen wird.

Der mehrstufige Verarbeitungstyp, der einer Treppe ähnelt, ermöglicht eine bequeme Verarbeitung von einer variablen Anzahl von Argumenten.

9.11.4 Zugriff auf Argumente

Um auf Argumente zuzugreifen, muss es für jedes Argument einen klar definierten Typ geben. Durch die extrem dynamische Art von PHP, müssen Sie auch hier auf einige Eigenarten achten. Da PHP niemals den Typ überprüft, kann ein Anrufer jede Art von Daten an Ihre Funktionen übergeben, unabhängig davon, ob Sie dies wollen oder nicht. Wenn Sie beispielsweise einen Integer erwarten, könnte der Anrufer einen Array übergeben oder umgekehrt. PHP wird es nicht bemerken.

Um dies zu umgehen, stehen Ihnen einige API-Funktionen zur Verfügung, bei denen Sie bei jedem Argument, das übergeben wurde, eine Typenkonvertierung erzwingen können (siehe Tabelle 9.4).

Hinweis: Alle Konvertierungsfunktionen erwarten als Parameter ein `**zval`.

Funktion	**Beschreibung**
`convert_to_boolean_ex(`*Wert*`)`	Erzwingt die Konvertierung in einen booleschen Wert. Boolesche Werte bleiben unberührt. Longs, Doubles und Strings, die `0` enthalten, sowie `NULL`-Werte ergeben den booleschen Wert `0` (`FALSE`). Arrays und Objekte werden, je nach der Anzahl der Einträge oder Parameter konvertiert, die sie haben. Leere Arrays und Objekte werden in `FALSE` umgewandelt, ansonsten in `TRUE`. Alle anderen Werte führen zum booleschen Wert `1` (`TRUE`).
`convert_to_long_ex(`*Wert*`)`	Erzwingt die Konvertierung in einen Long den Standard-Integertyp. `NULL`-Werte, boolesche Werte, Ressourcen und natürlich Longs bleiben unverändert. Doubles werden gekappt. Strings, die einen Integer enthalten, werden in die entsprechende numerische Darstellung umgewandelt, ansonsten in `0`. Arrays und Objekte werden in `0` konvertiert, sofern sie leer sind, ansonsten in `1`.
`convert_to_double_ex(`*Wert*`)`	Erzwingt die Konvertierung in einen Double, den Standardfließkommatyp. `NULL`-Werte, Boolesche Werte, Ressourcen, Longs und natürlich Doubles bleiben unverändert. Strings, die eine Zahl enthalten, werden in die entsprechende numerische Darstellung umgewandelt, ansonsten in `0.0`. Arrays und Objekte werden in `0.0` konvertiert, wenn sie leer sind, ansonsten in `1.0`.
`convert_to_string_ex(`*Wert*`)`	Erzwingt die Konvertierung in einen Sting. Strings bleiben unverändert. `NULL`-Werte werden in einen leeren String umgewandelt. Der boolesche Wert `TRUE` wird in `"1"` umgewandelt, ansonsten wird ein leerer String erzeugt. Longs und Doubles werden die entsprechende Stringdarstellung konvertiert. Arrays werden in den String »`Array`« und Objekte in den String »`Object`« umgewandelt.

Tab. 9.4: Funktionen zur Argumentenkonvertierung

Funktion	Beschreibung
convert_to_array_ex(*Wert*)	Erzwingt die Konvertierung in einen Array. Arrays bleiben unverändert. Objekte werden in einen Array konvertiert, indem alle ihre Parameter der Array-Tabelle zugewiesen werden. Die Parameternamen werden als Schlüssel verwendet, die Parameterinhalte als Werte. NULL-Werte werden in einen leeren Array umgewandelt. Alle anderen Werte werden in ein Array umgewandelt, in dem der Quellwert in dem Element mit dem Schlüssel 0 abgelegt wird.
convert_to_object_ex(*Wert*)	Erzwingt die Konvertierung in ein Objekt. Objekte bleiben unverändert. NULL-Werte werden in ein leeres Objekt umgewandelt. Arrays werden in Objekte konvertiert, indem ihre Schlüssel als Parameter und ihre Werte als entsprechende Parameterinhalte des Objekts integriert werden. Alle anderen Typen werden zu einem Objekt mit dem Parameter scalar, der den entsprechenden Quellwert enthält.
convert_to_null_ex(*Wert*)	Erzwingt die Konvertierung in einen NULL-Wert, d.h. leer.

Tab. 9.4: Funktionen zur Argumentenkonvertierung

Hinweis: Eine Demonstration hierzu finden Sie in cross_conversion.php auf der beiliegenden CD-ROM. Abbildung 9.5 zeigt die Ausgabe.

Wenn Sie diese Funktionen auf Ihre Argumente anwenden, können Sie die Typen der Daten, die an Sie übergeben werden, sicher festlegen. Wenn der gelieferte Typ nicht dem benötigten Typ entspricht, erzwingt PHP im erzeugten Wert einen Dummy-Inhalt (leere Strings, Arrays oder Objekte, 0 für numerische Werte, FALSE für boolesche Werte), um einen definierten Status zu gewährleisten.

Der folgende Ausschnitt stammt aus dem bereits besprochenen Beispielmodul, das die Konvertierungsfunktionen nutzt:

```
zval **parameter;

if((ZEND_NUM_ARGS() != 1) || (zend_get_parameters_ex(1, &parameter) !=
SUCCESS))
{
    WRONG_PARAM_COUNT;
}
```

```
convert_to_long_ex(parameter);

RETURN_LONG((*parameter)->value.lval);
```

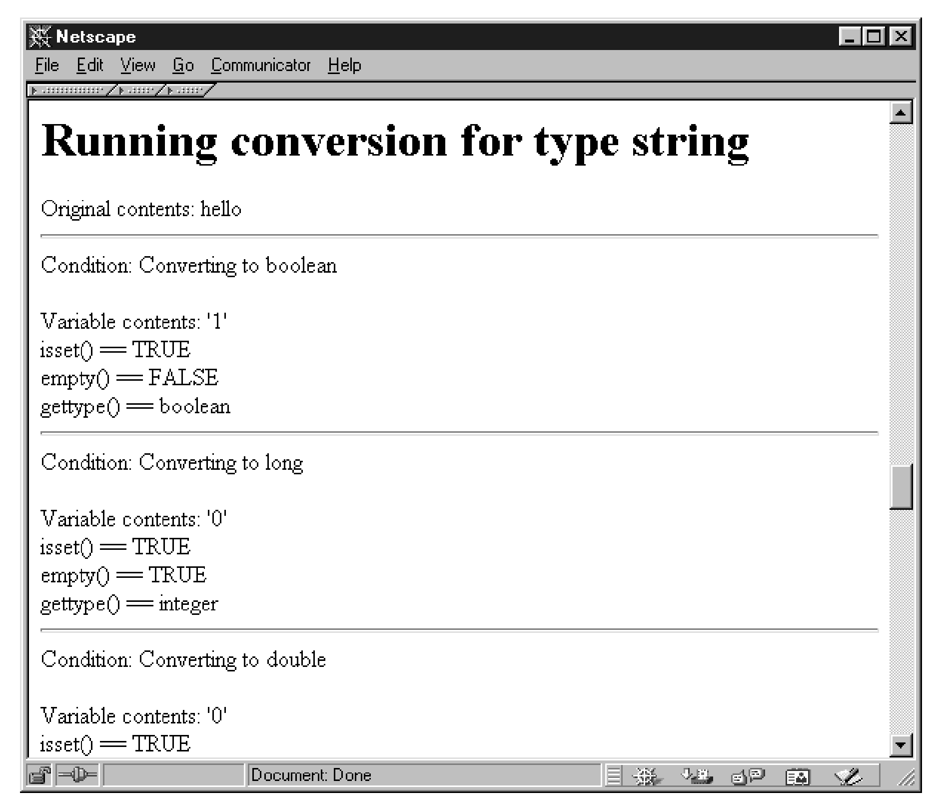

Abbildung 9.5: Cross-Conversion-Verhalten von PHP

Nach Anfrage des Parameterzeigers wird der Parameterwert in einen Long konvertiert (einen Integer), der auch den Rückgabewert dieser Funktion bildet. Um den Zugriff auf den Inhalt des Wertes zu verstehen, sollten wir uns kurz mit dem Typ `zval` befassen, dessen Definition Sie in Listing 9.8 sehen.

```
typedef pval zval;

typedef struct _zval_struct zval;

typedef union _zvalue_value {
   long lval;                /* long value */
   double dval;            /* double value */
   struct {
      char *val;
```

```
      int len;
   } str;
   HashTable *ht;            /* hash table value */
   struct {
      zend_class_entry *ce;
      HashTable *properties;
   } obj;
} zvalue_value;

struct _zval_struct {
   /* Variable information */
   zvalue_value value;       /* value */
   unsigned char type;    /* active type */
   unsigned char is_ref;
   short refcount;
};
```

Listing 9.8: Definition des PHP/Zend-Typs zval

Genau genommen ist pval (das in php.h definiert ist) nur ein Alias von zval (das in zend.h definiert ist), das seinerseits auf _zval_struct referenziert. Dies ist eine sehr interessante Struktur. _zval_struct ist die »Haupt«struktur, welche die Struktur des Wertes, den Typ und Referenzdaten enthält. Die Unterstruktur zvalue_value ist ein Verbund, der den Inhalt der Variablen enthält. Je nach dem Variablentyp müssen Sie auf verschiedene Mitglieder dieses Verbunds zugreifen. Eine Beschreibung der beiden Strukturen finden Sie in der Tabellen 9.5 bis 9.7.

Eintrag	Beschreibung
value	Verbund, der den Inhalt dieser Variablen enthält. Siehe Tabelle 9.6 für eine Beschreibung.
type	Enthält den Typ dieser Variablen. Siehe Tabelle 9.7 für eine Liste der verfügbaren Typen.
is_ref	0 bedeutet, dass diese Variable keine Referenz ist; 1 bedeutet, dass diese Variable eine Referenz auf eine andere Variable ist.
refcount	Anzahl der Referenzen, die für diese Variable existieren. Für jede neue Referenz auf diesen Wert, der in dieser Variablen gespeichert ist, wird der Zähler um 1 erhöht. Für jede verlorenen Referenz wird der Zähler um 1 heruntergesetzt. Wenn der Referenzzähler 0 erreicht, gibt es keine Referenzen auf diesen Wert mehr, was eine automatische Freigabe dieses Wertes zur Folge hat.

Tab. 9.5: Zend-Struktur zval

Eintrag	Beschreibung
lval	Verwenden Sie diesen Parameter, wenn die Variable vom Typ IS_LONG, IS_BOOLEAN oder IS_RESOURCE ist.
dval	Verwenden Sie diesen Parameter, wenn die Variable vom Typ IS_DOUBLE ist.
str	Diese Struktur kann verwendet werden, um auf Variablen des Typ IS_STRING zuzugreifen. Das Mitglied len enthält die Länge des Strings. Das Mitglied val zeigt auf den String selbst. Zend verwendet C-Strings. Die Längenangabe endet mit 0x00.
ht	Dieser Eintrag zeigt auf des Hash-Tabelleneintrag der Variablen, wenn die Variable ein Array ist.
obj	Verwenden Sie diesen Parameter , wenn die Variable vom Typ IS_OBJECT ist.

Tab. 9.6: Zend-Struktur zvalue_value

Konstante	Beschreibung
IS_NULL	ein NULL(leerer)-Wert
IS_LONG	ein Long(Integer)-Wert
IS_DOUBLE	ein Double(Fließkomma)-Wert
IS_STRING	ein String
IS_ARRAY	bezeichnet einen Array
IS_OBJECT	ein Objekt.
IS_BOOL	ein boolescher Wert
IS_RESOURCE	eine Ressource (zur Erörterung von Ressourcen siehe den entsprechenden Abschnitt weiter unten)
IS_CONSTANT	ein konstanter (definierter) Wert

Tab. 9.7: Zend-Variablentyp Konstante

Um auf einen Long zuzugreifen, verwenden Sie zval.value.lval, um auf eine Double zuzugreifen, verwenden sie zval.value.dval usw. Da alle Werte in einer Verbund gespeichert werden, führt der Versuch, auf Daten mit falschen Mitglieder des Verbunds zuzugreifen, zu einer sinnlosen Ausgabe.

Der Zugriff auf Arrays und Objekte ist etwas komplizierter und wird später erörtert.

9.11.5 Verarbeitung von Argumenten, die durch Referenz übergeben wurden

Wenn Ihre Funktion Argumente annimmt, die durch Referenz übergeben wurden und Sie diese ändern möchten, müssen Sie einige Vorkehrungen treffen.

Bisher nicht erwähnt haben wir, dass wir unter den bisher beschriebenen Umständen keinen Schreibzugriff auf `zval`-Container haben, die Funktionsparameter spezifizieren, die an Sie übergeben wurden. Natürlich können Sie jeden `zval`-Container ändern, den Sie innerhalb Ihrer Funktion erzeugt haben. Sie dürfen jedoch keine `zval`s ändern, die auf Zend-interne Daten referenzieren!

Wir haben bisher nur sogenannte `*_ex()`-API erörtert. Sie haben möglicherweise bemerkt, dass die API-Funktionen, die wir verwendet haben, nicht `zend_get_parameters()`, sondern `zend_get_parameters_ex()` bzw. nicht `convert_to_long()`, sondern `convert_to_long_ex()` usw. lauten. Die `*_ex()`-Funktionen bilden die sogenannten neue »erweiterte« Zend-API. Sie bieten eine leichte Geschwindigkeitserhöhung gegenüber der alten API, sind dafür aber nur für die Bereitstellung eines Nur-Lese-Zugriffs gedacht.

Da Zend intern mit Referenzen arbeitet, können verschiedene Variablen möglicherweise auf denselben Wert referenzieren. Der Schreibzugriff auf einen `zval`-Container setzt voraus, dass dieser Container einen separaten Wert enthält, d.h. auf einen Wert, auf den nicht von einem anderen Container referenziert wird. Wenn Sie mit anderen Containern auf den `zval`-Container referenzieren und das referenzierte `zval` ändern, müssen Sie automatisch den Inhalt der anderen Container ändern, die auf dieses `zval` referenzieren (da Sie einfach auf den geänderten Wert zeigen und daher auch Ihren eigenen Wert ändern würden).

`zend_get_parameters_ex()` kümmert sich nicht um diese Situation. Die Funktion gibt einfach einen Zeiger auf den gewünschten `zval`-Container aus, egal ob sie aus Referenzen bestehen oder nicht. Die entsprechende Funktion der traditionellen API `zend_get_parameters()` prüft sofort, ob referenzierte Werte vorliegen. Wenn sie eine Referenz findet, erzeugt sie einen neuen separaten `zval`-Container, kopiert die referenzierten Daten in diesen neu zugewiesenen Speicher und gibt einen Zeiger aus den neuen separaten Wert aus.

Dieser Vorgang `zval`-Separation (oder `pval`-Separation). Da die `*_ex()`-API keine `zval`-Separation durchführt, ist sie zwar um einiges schneller, verhindert aber einen Schreibzugriff.

Zum Ändern der Parameter benötigen Sie jedoch einen Schreibzugriff. Zend löst dieses Problem auf eine spezielle Weise: Sobald ein Parameter einer Funktion durch Referenz übergeben wird, wird automatisch eine `zval`-Separation

durchgeführt. Wenn Sie eine Funktion wie diese in PHP aufrufen, stellt Zend also automatisch sicher, dass `$parameter` als isolierter Wert übergeben wird, indem es ihn im schreibsicheren Status wiedergibt:[???]

```
my_function(&$parameter);
```

Dies ist jedoch nicht der Fall bei regulären Parametern! Alle anderen Parameter, die nicht durch Referenz übergeben werden, befinden sich in einem Nur-Lesen-Status.

Dazu müssen Sie jedoch sicherstellen, dass Sie wirklich mit einer Referenz arbeiten – ansonsten produzieren Sie möglicherweise unerwünschte Ergebnisse. Um einen Parameter zu suchen, der durch Referenz übergeben wurde, können Sie die Function `ParameterPassedByReference()` verwenden. Diese Funktion nimmt zwei Parameter an. Der erste ist der `ht`-Wert der Funktion und der zweite ist die Argumentenanzahl, die Sie testen möchten, wobei die Zählung von links nach rechts geht, wie Listing 9.9 und Abbildung 9.6 zeigen (den vollständigen Quellcode finden Sie auf der CD-ROM).

```
zval **parameter;

if((ZEND_NUM_ARGS() != 1) || (zend_get_parameters_ex(1, &parameter) !=
➥SUCCESS))
{
    WRONG_PARAM_COUNT;
}

/* check for parameter being passed by reference */
if(!ParameterPassedByReference(ht, 1))
{
    zend_error(E_WARNING, "Parameter wasn't passed by reference");
    RETURN_NULL();
}

/* make changes to the parameter */
(*parameter)->type = IS_LONG;
(*parameter)->value.lval = 10;
```

Listing 9.9: Austesten der Paramterübergabe durch Referenz

Netscape
File Edit View Go Communicator Help

Trying without reference operator

Before

Type: string
Content: hello

Warning: Parameter wasn't passed by reference in **/home/www/htdocs/reference_test.php** on line **18**

After

Type: string
Content: hello

Trying with reference operator

Before

Type: string
Content: hello

After

Type: integer
Content: 10

Document: Done

Abbildung 9.6: Austesten der Parameterübergabe durch Referenz

9.11.6 Gewährleistung der Schreibsicherheit für andere Parameter

Möglicherweise kommen Sie in eine Situation, in der Sie Schreibzugriff für einen Parameter benötigen, der mit `zend_get_parameters_ex()` ausgelesen wurde, aber nicht durch Referenz übergeben wurde. Für diesen Fall können Sie das Makro `SEPARATE_ZVAL` verwenden, das eine `zval`-Separation in bereitgestellte Container durchführt. Der neu erzeugte `zval` wird von den internen Daten abgetrennt und hat nur einen lokalen Gültigkeitsbereich. Damit kann er geändert oder entfernt werden, ohne dass dies Auswirkungen auf die globale Umgebung des Skripts hätte:

```
zval **parameter;

/* retrieve parameter */
zend_get_parameters_ex(1, &parameter);

/* at this stage, <parameter> still is connected */
/* to Zend's internal data buffers */

/* make <parameter> write-safe */
SEPARATE_ZVAL(parameter);

/* now we can safely modify <parameter> */
/* without implying global changes
*/
```

`SEPARATE_ZVAL` verwendet `emalloc()` zur Zuweisung des neuen `zval`-Containers; dies bedeutet, dass er automatisch gelöscht wird, wenn das Skript beendet wird, selbst wenn Sie die Zuweisung des Speichers nicht selbst aufheben. Wenn Sie dieses Makro jedoch häufig aufrufen, ohne die entfernten Container wieder freizugeben, füllen Sie langsam Ihr RAM.

> **Hinweis:** Da Sie den Mangel an Schreibsicherheit bei der »traditionellen« API umgehen können (mit `zend_get_parameters()` usw.), scheint diese API veraltet zu sein. Deshalb werden in diesem Kapitel nicht mehr darauf eingehen.

9.12 Erzeugen von Variablen

Wenn Sie Daten von Ihren eigenen Erweiterungen mit PHP-Skripten austauschen, betrifft eine der wichtigsten Fragen die Erzeugung von Variablen. Dieser Abschnitt beschreibt die von PHP unterstützten Variablentypen.

9.12.1 Überblick

Um neue Variablen zu generieren, die vom ausführenden Skript »von außen« gesehen werden können, müssen Sie einen neuen `zval`-Container zuordnen, diesen Container mit sinnvollen Daten füllen und ihn anschließend in die interne Symboltabelle von Zend einfügen. Diese Vorgehensweise gilt für die Erzeugung aller Variablen:

```
zval *new_variable;

/* allocate and initialize new container */
MAKE_STD_ZVAL(new_variable);
```

```
/* set type and variable contents here, see the following sections */

/* introduce this variable by the name "new_variable_name" into the symbol
➥table */
ZEND_SET_SYMBOL(EG(active_symbol_table), "new_variable_name", new_
➥variable);

/* the variable is now accessible to the script by using $new_variable_name
*/
```

Das Makro MAKE_STD_ZVAL weist mithilfe von ALLOC_ZVAL einen neuen zval-Container zu und initialisiert ihn mit INIT_ZVAL. Gemäß der Realisierung in Zend zum Zeitpunkt der Bucherstellung bedeutet initializing, dass der Referenzzähler auf 1 gesetzt wird und der Flag is_ref gelöscht wird. Dieser Prozess wird jedoch später erweitert werden. Deshalb sollten Sie nicht nur ALLOC_ZVAL verwenden, sondern auch MAKE_STD_ZVAL im Hinterkopf behalten. Wenn Sie die Geschwindigkeit optimieren möchten, müssen Sie hier nicht unbedingt den zval-Container initialisieren. Sie können ALLOC_ZVAL verwenden, was wir jedoch nicht empfehlen, da es keine Datenintegrität gewährleistet.

ZEND_SET_SYMBOL trägt die neue Variable in die Zend-Symboltabelle ein. Dieses Makro überprüft, ob der Wert bereits in der Symboltabelle vorhanden ist, und konvertiert das neue Symbol in eine Referenz, wenn dies der Fall ist und hebt die Zuweisung des alten zval-Containers automatisch auf. Diese Methode wird bevorzugt, wenn Geschwindigkeit kein ausschlaggebender Faktor ist und Sie die Speicherplatzbelegung gering halten möchten.

Beachten Sie, dass ZEND_SET_SYMBOL über das Makro EG globale Variablen den Zend-Executors nutzt. Durch Angabe von EG(active_symbol_table) können Sie auf die derzeit aktive Symboltabelle zugreifen, die sich auf den aktiven lokalen Gültigkeitsbereich bezieht. Je nachdem ob die Funktion innerhalb einer Funktion aufgerufen wurde, kann der lokale Gültigkeitsbereich variieren.

Wenn Sie die Geschwindigkeit optimieren möchten und Ihnen eine optimale Speicherbelegung nicht wichtig ist, können Sie die Überprüfung auf existierende Variablen mit demselben Wert weglassen und mithilfe von zend_hash_update() ein Einfügen der Variablen in die Symboltabelle erzwingen:

```
zval *new_variable;

/* allocate and initialize new container */
MAKE_STD_ZVAL(new_variable);

/* set type and variable contents here, see the following sections */

/* introduce this variable by the name "new_variable_name" into the symbol
➥table */
```

```
zend_hash_update(EG(active_symbol_table), "new_variable_name",
strlen("new_variable_name") + 1, &new_variable, sizeof(zval *), NULL);
```

Tatsächlich ist dies die Standardmethode, welche die meisten Module verwenden.

Die mit dem obigen Code erzeugte Variablen, haben stets nur lokal Gültigkeit, so dass sie in dem Kontext abgelegt werden, in dem die Funktion aufgerufen wurde. Um neue Variablen auf globaler Ebene zu erzeugen, verwenden Sie dieselbe Methode, referenzieren aber auf eine andere Symboltabelle:

```
zval *new_variable;

// allocate and initialize new container
MAKE_STD_ZVAL(new_variable);

//
// set type and variable contents here
//

// introduce this variable by the name "new_variable_name" into the global
symbol table
ZEND_SET_SYMBOL(&EG(symbol_table), new_variable);
```

Das Makro `ZEND_SET_SYMBOL` wird nun mit einer Referenz auf die globale Hauptsymboltabelle aufgerufen, indem `EG(symbol_table)` referenziert wird.

Hinweis: Die Variable `active_symbol_table` ist ein Zeiger, `symbol_table` hingegen nicht. Deswegen müssen Sie `EG(active_symbol_table)` und `&EG(symbol_table)` als Parameter zu `ZEND_SET_SYMBOL` verwenden; dieses Makro benötigt einen Zeiger.

Wenn Sie die Version noch effizienter machen wollen, können Sie die Aktualisierung der Symboltabelle auch direkt im Programm kodieren:

```
zval *new_variable;

// allocate and initialize new container
MAKE_STD_ZVAL(new_variable);

//
// set type and variable contents here
//

// introduce this variable by the name "new_variable_name" into the global
➥symbol table
zend_hash_update(&EG(symbol_table), "new_variable_name",
strlen("new_variable_name") + 1, &new_variable, sizeof(zval *), NULL);
```

Listing 9.10 zeigt einen Quellcode, der zwei Variablen erzeugt: `local_variable` mit lokalem Gültigkeitsbereich und `global_variable` mit einem globalen Gültigkeitsbereich (siehe Abbildung 9.7). Das vollständige Beispiel finden Sie auf der CD-ROM.

Netscape
File Edit View Go Communicator Help

Creating variables in a local scope...

Local Variable

Type: integer
Contents: 10

Global Variable

Type: NULL
Contents:

Accessing variables from the main scope...

Local Variable

Type: NULL
Contents:

Global Variable

Type: integer
Contents: 5

Document: Done

Abbildung 9.7: Variablen mit verschiedenen Gültigkeitsbereichen

Hinweis: Wie Sie sehen, kann auf die globale Variable nicht innerhalb der Funktion zugegriffen werden. Dies liegt darin, dass sie im PHP-Quellcode nicht mit `global $global_variable;` in den lokalen Gültigkeitsbereich importiert wurde.

```
ZEND_FUNCTION(variable_creation)
{
    zval *new_var1, *new_var2;

    MAKE_STD_ZVAL(new_var1);
    MAKE_STD_ZVAL(new_var2);

    new_var1->type = IS_LONG;
    new_var1->value.lval = 10;

    new_var2->type = IS_LONG;
    new_var2->value.lval = 5;

    ZEND_SET_SYMBOL(EG(active_symbol_table), "local_variable", new_var1);
    ZEND_SET_SYMBOL(&EG(symbol_table), "global_variable", new_var2);

    RETURN_NULL();

}
```

Listing 9.10: Erzeugung von Variablen mit verschiedenen Gültigkeitsbereichen

9.12.2 Longs (Integer)

Befassen wir uns nun mit der Zuordnung von Daten zu Variablen. Wir beginnen mit Longs. Longs sind die Integer von PHP und sehr einfach zu speichern. Wenn Sie sich noch einmal die besprochene Containerstruktur `zval.value` ansehen, können Sie feststellen, dass der Datentyp Long direkt im Verbund enthalten ist, und zwar im Feld `lval`. Der entsprechende `type`-Wert für Longs ist `IS_LONG` (siehe Listing 9.11).

```
zval *new_long;

MAKE_STD_ZVAL(new_long);

new_long->type = IS_LONG;
new_long->value.lval = 10;
```

Alternativ können Sie das Makro `ZVAL_LONG` verwenden:

```
zval *new_long;

MAKE_STD_ZVAL(new_long);
ZVAL_LONG(new_long, 10);
```

Listing 9.11: Generierung eines Longs

9.12.3 Doubles (Fließkommazahlen)

Doubles sind die Fließkommazahlen von PHP, und genauso einfach zuzuweisen wie Longs. Auch ihr Wert ist direkt im Verbund enthalten. Das entsprechende Mitglied im `zval.value`-Container heißt `dval`; der zugehörige Typ ist `IS_DOUBLE`.

```
zval *new_double;

MAKE_STD_ZVAL(new_double);

new_double->type = IS_DOUBLE;
new_double->value.dval = 3.45;
```

Alternativ können Sie das Makro `ZVAL_DOUBLE` verwenden.

```
zval *new_double;

MAKE_STD_ZVAL(new_double);
ZVAL_DOUBLE(new_double, 3.45);
```

9.12.4 Strings

Strings sind etwas aufwendiger. Wie zuvor bereits erwähnt, müssen alle Strings, die mit den internen Datenstrukturen von Zend verknüpft werden sollen, über Zends eigene Speicherverwaltungsfunktionen zugewiesen werden. Das Referenzieren von statischen Strings oder Strings, die mit Standardroutinen zugewiesen wurden, ist nicht zulässig. Um Strings zuzuweisen, müssen Sie auf die Struktur `str` im `zval.value`-Container zugreifen. Der entsprechende Typ lautet `IS_STRING`:

```
zval *new_string;
char *string_contents = "This is a new string variable";

MAKE_STD_ZVAL(new_string);

new_string->type = IS_STRING;
new_string->value.str.len = strlen(string_contents);
new_string->value.str.val = estrdup(string_contents);
```

Beachten Sie hier die Verwendung von Zends `estrdup()`. Alternativ können Sie auch das vordefinierte Makro `ZVAL_STRING` verwenden:

```
zval *new_string;
char *string_contents = "This is a new string variable";

MAKE_STD_ZVAL(new_string);
ZVAL_STRING(new_string, string_contents, 1);
```

ZVAL_STRING besitzt einen dritten Parameter; dieser gibt an, ob der bereitgestellte Stringinhalt dupliziert werden soll (mithilfe von estrdup()). Durch Setzen dieses Parameters auf 1 wird der String dupliziert, bei 0 wird lediglich der bereitgestellte Zeiger für den Variableninhalt verwendet. Dieses Makro ist insbesondere dann nützlich, wenn Sie eine neue Variable erzeugen möchten, indem Sie auf einen String referenzieren, der bereits im internen Speicher von Zend zugewiesen wurde.

Wenn Sie den String an einer bestimmten Stelle kappen möchten oder seine Länge bereits im voraus kennen, können Sie ZVAL_STRINGL(zval, string, length, duplicate) verwenden. Dieses Makro lässt die Angabe einer exakten Länge für den neuen String zu und ist schneller als ZVAL_STRING und außerdem binärsicher.

Zum Generieren eines leeren Strings, setzen Sie die Stringlänge auf 0 und verwenden Sie als Inhalt empty_string:

```
new_string->type = IS_STRING;
new_string->value.str.len = 0;
new_string->value.str.val = empty_string;
```

Natürlich gibt es auch hierfür ein Makro (ZVAL_EMPTY_STRING):

```
MAKE_STD_ZVAL(new_string);
ZVAL_EMPTY_STRING(new_string);
```

9.12.5 Boolesche Werte

Boolesche Werte werden genauso erzeugt, wie Longs, haben aber den Typ IS_BOOL. Zulässige Werte für lval sind 0 und 1:

```
zval *new_bool;

MAKE_STD_ZVAL(new_bool);

new_bool->type = IS_BOOL;
new_bool->value.lval = 1;
```

Die entsprechende Makros für diesen Typ heißen ZVAL_BOOL (das eine Wertangabe ermöglicht) sowie ZVAL_TRUE und ZVAL_FALSE (welche den Wert explizit auf TRUE bzw. FALSE setzen).

9.12.6 Arrays

Arrays werden in den internen Hash-Tabellen von Zend gespeichert, auf die Sie über die API zend_hash_*() zugreifen können. Für jedes Array, das Sie erzeugen möchten, benötigen Sie einen neuen Hash-Tabellen-Handle, der im Mitglied ht des zval.value-Containers gespeichert wird.

Es gibt eine API, die nur für die Erstellung von Arrays zuständig ist. Dies ist sehr praktisch. Um ein neues Array zu initialisieren, rufen Sie `array_init()` auf:

```
zval *new_array;

MAKE_STD_ZVAL(new_array);

if(array_init(new_array) != SUCCESS)
{
    // do error handling here
}
```

Wenn `array_init()` kein neues Array generieren kann, wird `FAILURE` zurückgegeben.

Zum Hinzufügen neuer Elemente in das Array können Sie zahlreiche Funktionen verwenden, je nachdem, was Sie tun möchten. In den Tabellen 9.8 bis 9.10 erhalten Sie eine Beschreibung dieser Funktionen. Alle Funktionen geben beim Fehlschlagen `FAILURE` zurück, im Falle eines Erfolgs `SUCCESS`.

Hinweis: Die Funktionen aus Tabelle 9.8 arbeiten alle mit dem Array `array` mit dem Schlüssel `key`. Der Schlüselstring muss sich nicht im internen Zendspeicher befinden; er wird von der API dupliziert.

Funktion	Beschreibung
`add_assoc_long(zval *array, char *key, long n)`	Fügt ein Element des Typs `long` hinzu.
`add_assoc_unset(zval *array, char *key)`	Fügt ein nicht gesetztes Element hinzu.
`add_assoc_bool(zval *array, char *key, int b)`	Fügt ein boolesches Element hinzu.
`add_assoc_resource(zval *array, char *key, int r)`	Fügt im Array eine Ressource hinzu.
`add_assoc_double(zval *array, char *key, double d)`	Fügt eine Fließkommazahl hinzu.
`add_assoc_string(zval *array, char *key, char *str, int duplicate)`	Fügt im Array einen String hinzu. Der Flag `duplicate` gibt an, ob der Stringinhalt in den internen Zendspeicher kopiert werden muss.
`add_assoc_stringl(zval *array, char *key, char *str, uint length, int duplicate)`	Fügt einen String der gewünschten Länge `length` im Array hinzu, ansonsten verhält sich diese Funktion wie `add_assoc_string()`.

Tab. 9.8: Zends API für assoziative Arrays

Hinweis: Die Funktionen aus Tabelle 9.9 arbeiten mit dem Array `array` mit dem Index `idx`. Der Index ist stets ein Integer.

Funktion	Beschreibung
`add_index_long(zval *array, uint idx, long n)`	Fügt ein Element des Typs `long` hinzu.
`add_index_unset(zval *array, uint idx)`	Fügt ein nicht gesetztes Element hinzu.
`add_index_bool(zval *array, uint idx, int b)`	Fügt ein boolesches Element hinzu.
`add_index_resource(zval *array, uint idx, int r)`	Fügt im Array eine Ressource hinzu.
`add_index_double(zval *array, uint idx, double d)`	Fügt eine Fließkommazahl hinzu.
`add_index_string(zval *array, uint idx, char *str, int duplicate)`	Fügt im Array einen String hinzu. Der Flag `duplicate` gibt an, ob der Stringinhalt in den internen Zend-Speicher kopiert werden muss.
`add_index_stringl(zval *array, uint idx, char *str, uint length, int duplicate)`	Fügt im Array einen String der gewünschten Länge `length` hinzu. diese Funktion ist schneller und binärsicher. Ansonsten verhält sie sich wie `add_index_string()`.

Tab. 9.9: Zends API für indizierte Arrays, Teil 1

Hinweis: Die Funktionen aus der Tabelle 9.10 arbeiten mit dem Array `array`. Diese Funktionen generieren automatisch einen neuen Index, basierend auf dem höchsten, im Array gefundenen, Index.

Funktion	Beschreibung
`add_next_index_long(zval *array, long n)`	Fügt ein Element des Typs `long` hinzu.
`add_next_index_unset(zval *array);`	Fügt ein nicht gesetztes Element hinzu.
`add_next_index_bool(zval *array, int b)`	Fügt ein boolesches Element hinzu.
`add_next_index_resource(zval *array, int r)`	Fügt im Array eine Ressource hinzu.

Tab. 9.10: Zends API für indexierte Arrays, Teil 2

Funktion	Beschreibung
`add_next_index_double(zval *array, double d)`	Fügt eine Fließkommazahl hinzu.
`add_next_index_string(zval *array, char *str, int duplicate)`	Fügt im Array einen String hinzu. Der Flag `duplicate` gibt an, ob der Stringinhalt in den internen Zend-Speicher kopiert werden muss.
`add_next_index_stringl(zval *array, char *str, uint length, int duplicate)`	Fügt im Array einen String der gewünschten Länge `length`. Diese Funktion ist schneller und binärsicher. Ansonsten verhält sie sich wie `add_index_string()`.

Tab. 9.10: Zends API für indexierte Arrays, Teil 2

Diese Funktionen bieten alle eine praktische Abstrahierung der internen Hash-API von Zend. Natürlich können Sie die Hash-Funktionen auch direkt verwenden, z.B. wenn Sie bereits einen `zval`-Container zugewiesen haben, den Sie in einen Array einfügen möchten. Bei assoziativen Arrays geschieht dies über `zend_hash_update()` (siehe Listing 9.12) und bei indizierten Arrays über `zend_hash_index_update()` (siehe Listing 9.13):

```
zval *new_array, *new_element;
char *key = "element_key";

MAKE_STD_ZVAL(new_array);
MAKE_STD_ZVAL(new_element);

if(array_init(new_array) == FAILURE)
{
    // do error handling here
}

ZVAL_LONG(new_element, 10);

if(zend_hash_update(new_array->value.ht, key, strlen(key) + 1,
➥(void *)&new_element, sizeof(zval *), NULL) == FAILURE)
{
    // do error handling here
}
```

Listing 9.12: Hinzufügen eines Elements in einen assoziativen Array

```
zval *new_array, *new_element;
int key = 2;

MAKE_STD_ZVAL(new_array);
MAKE_STD_ZVAL(new_element);
```

```
if(array_init(new_array) == FAILURE)
{
    // do error handling here
}

ZVAL_LONG(new_element, 10);

if(zend_hash_index_update(new_array->value.ht, key, (void *)&new_element,
➥sizeof(zval *), NULL) == FAILURE)
{
    // do error handling here
}
```

Listing 9.13: Hinzufügen eines Elements in einen indizierten Array

Um die Funktionalität von `add_next_index_*()` zu emulieren, können Sie folgende Programmzeile verwenden:

```
zend_hash_next_index_insert(ht, zval **new_element, sizeof(zval *), NULL)
```

Hinweis: Um mit einer Funktion einen Array zurückzugeben, verwenden Sie `array_init()` und für die folgenden Aktionen die vordefinierte Variable `return_value` (die als Argument Ihrer exportierten Funktion angegeben wird; siehe hierzu die frühere Erörterung der Aufrufschnittstelle). Sie müssen hierfür nicht `MAKE_STD_ZVAL` verwenden.

Tipp: Um zu vermeiden, dass Sie jedes Mal `new_array->value.ht` schreiben müssen, können Sie `HASH_OF(new_array)` verwenden; dies ist auch aus Kompatibilitätsgründen empfehlenswert und zeugt von gutem Programmierstil.

9.13 Objekte

Da Objekte in Arrays konvertiert werden können (und umgekehrt), haben Sie möglicherweise schon vermutet, dass Objekte in PHP große Ähnlichkeit mit Arrays haben. Objekte werden mit denselben Hash-Funktionen gepflegt, die API zur Generierung von Objekten ist jedoch eine andere.

Um ein Objekt zu initialisieren, verwenden Sie die Funktion `object_init()`:

```
zval *new_object;

MAKE_STD_ZVAL(new_object);

if(object_init(new_object) != SUCCESS)
{
```

```
    // do error handling here
}
```

Sie können die in Tabelle 9.11 beschriebenen Funktionen verwenden, um Mitglieder zu Ihrem Objekt hinzuzufügen.

Hinweis: Alle Funktionen aus Tabelle 9.11 arbeiten mit dem Objekt `object` und dem Schlüssel `key`. Der Schlüssel bildet den Namen des Mitglieds, so dass auf das erzeugte Mitglied über `$object->key` zugegriffen werden kann.

Funktion	Beschreibung
`add_property_long(zval *object, char *key, long l)`	Fügt einen Long zum Objekt hinzu.
`add_property_unset(zval *object, char *key)`	Fügt einen nicht gesetzten Parameter zum Objekt hinzu.
`add_property_bool(zval *object, char *key, int b)`	Fügt einen booleschen Wert zum Objekt hinzu.
`add_property_resource(zval *object, char *key, int r)`	Fügt eine Ressource zum Objekt hinzu.
`add_property_double(zval *object, char *key, double d)`	Fügt einen Double zum Objekt hinzu.
`add_property_string(zval *object, char *key, char *str, int duplicate)`	Fügt einen String zum Objekt hinzu.
`add_property_stringl(zval *object,lchar *key, char *str, uint length, int duplicate)`	Fügt einen String der angegebenen Länge zum Objekt hinzu. Diese Funktion ist schneller als `add_property_string` und außerdem binärsicher.

Tab. 9.11: Zends API für die Erzeugung von Objekten

9.14 Ressourcen

Ressourcen sind eine spezielle Art von Datentyp in PHP. Der Begriff *Ressource* bezieht sich nicht auf eine spezielle Art von Daten, sondern vielmehr auf eine Abstraktionsmethode zur Pflege beliebiger Informationen. Ressourcen werden innerhalb von Zend in speziellen Ressourcenlisten abgelegt. Jeder Eintrag in der Liste hat eine entsprechende Typdefinition, welche die Art von Ressource angibt, auf die sie referenziert. Intern verwaltet Zend alle Referenzen auf diese Ressource. Der Zugriff auf eine Ressource erfolgt niemals direkt sondern immer über die bereitgestellte API. Sobald es keine Referenzen auf eine bestimmte Ressource mehr gibt, wird eine entsprechende Deinitialisierungsfunktion aufgerufen.

Ressourcen werden beispielsweise verwendet, um Datenbanklinks und Dateideskriptoren zu speichern. Die *de facto* -Standardimplementierung finden Sie im MySQL-Modul. Aber auch andere Module, wie etwa das Oracle-Modul, verwenden Ressourcen.

Um einen Handle für Ihre spezielle Ressource zu erhalten, müssen Sie den Ressourcentyp vor der Verwendung registrieren:

```
int resource_handle = register_list_destructors(destructor_handler, NULL);
```

Dieser Aufruf erzeugt einen Handle, den Sie verwenden können, wenn Sie Einträge in die Ressourcenliste hinzufügen. Die angegebene Funktion (hier `destructor_handler`) wird stets aufgerufen, wenn es keine Referenz mehr auf eine bestimmte Ressource gibt und Zend versucht, diese zu löschen. Diese Funktion sorgt für die richtige Ressourcenfreigabe und Aufhebung der Zuweisungen. Sie muss vom Typ `void` sein, und als einziges Argument einen Zeiger auf den Typ besitzen, den Sie in die Liste einfügen möchten.

```
typedef struct
{
    int resource_link;
    int resource_type;
} my_resource;

void destructor_handler(my_resource *resource)
{

    // do all deallocation relating to the resource here

    // free container
    efree(resource);

}
```

Um nun eine Ressource zur Liste hinzuzufügen, verwenden Sie `zend_list_insert()`:

```
my_resource *resource;

// allocate resource here and fill it with values
resource = (my_resource *)emalloc(sizeof(my_resource));

resource_value = zend_list_insert(resource, resource_handle);
```

Die Funktion hat zwei Argumente. Das erste ist der Zeiger auf die Ressource, die Sie in die Liste einfügen möchten. Das zweite gibt den Typ der Ressource an, für den Sie zuvor einen Destruktor registriert haben. Anschließend können Sie den Rückgabewert des `zend_list_insert()`-Aufrufs im `value`-Feld des entsprechenden `zval`-Containers `IS_RESOURCE` verwenden.

Um beispielsweise die zugewiesene Ressource als Rückgabewert zu verwenden, geben Sie Folgendes ein:

```
RETURN_RESOURCE(resource_value);
```

Oder etwas eleganter:

```
return_value->type = IS_RESOURCE;
return_value->value.lval = resource_value;
```

Wie Sie sehen, werden die Ressorcen im Feld `lval` gespeichert.

Nun verfolgt Zend alle Referenzen auf diese Ressource. Sobald es keine Referenzen auf diese Ressource mehr gibt, wird der Destruktor aufgerufen, den Sie zuvor für diese Ressource registriert haben. Der schöne Nebeneffekt dieser Einstellung ist, dass Sie sich nicht um Speicherlöcher kümmern müssen, die durch Zuweisungen in Ihrem Modul entstehen. Sie registrieren einfach alle Speicherzuweisungen als Ressourcen, auf die Ihr aufrufendes Skript referenziert. Sobald das Skript beschließt, dass es diese nicht mehr benötigt, stellt Zend dieses fest und informiert Sie.

Um eine Ressource explizit aus einer Liste zu entfernen, verwenden Sie die Fuktion `zend_list_delete()`. Des weiteren können Sie auch die Erhöhung des Referenzzählers erzwingen, wenn Sie genau wissen, dass Sie eine weitere Referenz für einen bereits zugewiesenen Wert generieren (z.B. wenn Sie einen Standarddatenbanklink wiederverwenden). Für diesen Fall verwenden Sie die Funktion `zend_list_addref()`. Um bereits zugewiesene Ressourceneinträge zu suchen, verwenden Sie `zend_list_find()`. Die vollständige API finden Sie in `zend_list.h`.

Ein kleines Beispiel, das zeigt, wie Sie Ressourcen nutzen, finden Sie auf der CD-ROM.

9.15 Makros zur automatischen Erstellung von globalen Variablen

Neben den bereits besprochenen Makros gibt es einige Makros, mit denen wir auf einfache Weise globale Variablen erstellen können. Dies kann sehr nützlich sein, wenn wir z.B. globale Flags einführen wollen. In der Praxis wird dies zwar nicht gern gesehen, aber Tabelle 9.12 beschreibt Makros, die genau dies tun. Sie benötigen keine `zval`-Zuordnung. Sie brauchen lediglich einen Variablennamen und einen Wert.

> **Hinweis:** Alle Makros aus Tabelle 9.12 erzeugen eine globale Variable des Namens `name` mit dem Wert `value`.

Makro	Beschreibung
`SET_VAR_STRING(name, value)`	Erzeugt einen neuen String.
`SET_VAR_STRINGL(name, value, length)`	Erzeugt einen neuen String der angegebenen Länge. Dieses Makro ist schneller als `SET_VAR_STRING` und außerdem binärsicher.
`SET_VAR_LONG(name, value)`	Erzeugt einen neuen Long.
`SET_VAR_DOUBLE(name, value)`	Erzeugt einen neuen Double.

Tab. 9.12: Makros zur Erstellung globaler Variablen

9.15.1 Erstellen von Konstanten

Zend unterstützt die Erstellung von echten Konstanten (im Gegensatz zu regulären Variablen). Der Zugriff auf Konstanten erfolgt ohne das typische Dollarzeichen(`$`)-Präfix. Konstanten sind sowohl lokal als auch global verfügbar. Zwei Beispiele für Konstanten sind `TRUE` und `FALSE`.

Mit den Makros aus Tabelle 9.13 können Sie eigene Konstanten erzeugen. Diese Makros generieren eine Konstante mit dem angegebenen Namen und Wert.

Für jede Konstante können Sie auch Flags angeben:

- `CONST_CS` - Bei dem Namen dieser Konstante wird Groß- und Kleinschreibung unterschieden.
- `CONST_PERSISTENT` - Diese Konstante ist beständig und wird nicht »vergessen«, wenn der aktuelle Prozess, der diese Konstante enthält, beendet wird.

Um die Flags zu verwenden, kombinieren Sie diese mit einem binären ODER:

```
// register a new constant of type "long"
REGISTER_LONG_CONSTANT("NEW_MEANINGFUL_CONSTANT", 324, CONST_CS |
➥CONST_PERSISTENT);
```

Es gibt zwei Arten von Makros – `REGISTER_*_CONSTANT` und `REGISTER_MAIN_*_CONSTANT`. Der erste Typ erzeugt Konstanten, die an das aktuelle Modul gebunden sind. Diese Konstanten werden aus der Symboltabelle entfernt, sobald das Modul, das diese Konstante registriert hat, aus dem Speicher gelöscht wird. Der zweite Typ erzeugt Konstanten, die unabhängig vom Modul in der Symboltabelle verbleiben.

Makro	Beschreibung
REGISTER_LONG_CONSTANT(*Name, Wert, Flags*) REGISTER_MAIN_LONG_CONSTANT(*Name, Wert, Flags*)	Registriert eine neue Konstante des Typs long.
REGISTER_DOUBLE_CONSTANT(*Name, Wert, Flags)* REGISTER_MAIN_DOUBLE_CONSTANT(*Name, Wert, Flags*)	Registriert eine neue Konstante des Typs Double
REGISTER_STRING_CONSTANT(*Name, Wert, Flags*) REGISTER_MAIN_STRING_CONSTANT(*Name, Wert, Flags*)	Registriert eine neue Konstante des Typs String. Der angegeben String muss sich im internen Zend-Speicher befinden.
REGISTER_STRINGL_CONSTANT(*Name, Wert, Flags*) REGISTER_MAIN_STRINGL_CONSTANT(*Name, Wert, Länge, Flags*)	Registriert eine neue Konstante des Typs String. Die Stringlänge ist explizit auf length gesetzt. Der angegebene String muss sich im internen Zend-Speicher befinden.

Tab. 9.13: Makros zur Erstellen von Konstanten

9.16 Duplizierung variabler Inhalte: Der Copy-Konstruktor

Früher oder später müssen Sie möglicherweise den Inhalt eines zval-Containers einem anderen zuweisen. Dies ist leichter gesagt als getan, denn zval-Container enthalten nicht nur Informationen über Typen sondern auch Referenzen auf Bereiche in den internen Daten von Zend. Je nach Größe können Arrays oder Objekte mit vielen Hash-Tabelleneinträgen verschachtelt werden. Durch Zuweisung eines zval zu einem anderen, vermeiden Sie Duplizierungen der Hash-Tabelleneinträge, in denen Sie nur eine Referenz auf diese verwenden (wenn überhaupt).

Um diese komplexe Art von Daten zu kopieren, verwenden Sie den Copy-Konstruktor. Copy-Konstruktoren sind in der Regel in Sprachen definiert, die ein Überladen des Operators unterstützen und explizit dazu dienen, komplexe Typen zu kopieren. Wenn Sie in einer solchen Sprache ein Objekt definieren, haben Sie die Möglichkeit, den Operator = zu überladen, der in der Regel dazu verwendet wird, den Inhalt von lvalue (Ergebnis der Auswertung links vom Operator) rvalue (das Gleiche auf der rechten Seite) zuzuweisen.

Überladen bedeutet, dass dem Operator eine andere Bedeutung zugewiesen wird. In der Regel wird es verwendet, um einen Funktionsaufruf einem Operator zuzuordnen. Wenn dieser Operator dann auf solch ein Objekt in einem Programm angewendet würde, dann wird die Funktion mit den Parametern

lvalue und rvalue aufgerufen. Mit dieser Information ausgestattet, kann es die Operation ausführen, die der Operator = haben soll (in der Regel eine erweiterte Form des Kopierens).

Die gleiche Form des »erweiterten Kopierens« ist in PHP auch für die zval-Container erforderlich. Auch hier würde bei einem Array das erweiterte Kopieren die erneute Erstellung aller Hash-Tabelleneinträge beinhalten, die sich auf diesen Array beziehen. Bei Strings müsste eine korrekte Speicherzuweisung gewährleistet werden usw.

Zend verfügt über eine solche Funktion namens zend_copy_ctor() (das vorherige PHP-Äquivalent hieß pval_copy_constructor()).

Eine sehr schöne Demonstration für die Funktionsweise ist die folgende Funktion, die als Argument einen komplexen Typ hat, diesen modifiziert und anschließend das Argument zurückgibt:

```
zval **parameter;

if((ZEND_NUM_ARGS() != 1) || (zend_get_parameters_ex(1, &parameter) !=
➥SUCCESS))
{
    WRONG_PARAM_COUNT;
}

// do modifications to the parameter here

// now we want to return the modified container:
*return_value == **parameter;
zval_copy_ctor(return_value);
```

Der erste Teil der Funktion betrifft nur das reine Auslesen des Arguments. Nach den (hier ausgelassenen) Modifikationen, wird es jedoch interessant. Der Container von parameter wird dem (vordefinierten) return_value-Container zugeordnet. Um seinen Inhalt auf effektive Weise zu kopieren, wird der Copy-Konstruktor aufgerufen. Der Copy-Konstruktor arbeitet direkt mit dem bereitgestellten Argument, wobei standardmäßig die Rückgabewerte FAILURE (bei einer fehlgeschlagenen Aktion) bzw. SUCCESS (bei einer erfolgreichen Aktion) ausgegeben werden.

Wenn Sie in diesem Beispiel den Aufruf des Copy-Konstruktors weglassen, würden sowohl parameter als auch return_value auf dieselben internen Daten zeigen, was bedeutet, dass return_value eine nicht zulässige zusätzliche Referenz auf dieselben Datenstrukturen wäre. Sobald Änderungen in den Daten vorgenommen werden, auf welche parameter zeigt, könnte auch return_value betroffen sein. Um separate Kopien zu erstellen, müssen Sie demnach den Copy-Konstruktor verwenden.

Das Gegenstück des Copy-Konstruktors in der Zend-API ist der Destruktor `zval_dtor()`; er tut genau das Gegenteil vom Konstruktor. Der entsprechene Alias in der PHP-API lautet `pval_destructor()`.

9.17 Rückgabe von Werten

Auf die Rückgabe von Werten aus Funktionen in PHP sind wir in einem vorangegangenen Abschnitt bereits kurz eingegangen. In diesem Abschnitt erfahren Sie nähere Einzelheiten. Rückgabewerte werden durch die Variable `return_value` übergeben, die an Ihre Funktionen als Argument übergeben werden. Das Argument von `return_value` besteht aus einem `zval`-Container (siehe dazu die frühere Erörterung der Aufrufschnittstelle), den Sie frei verändern können. Der Container selbst ist bereits zugewiesen, so dass Sie für ihn `MAKE_STD_ZVAL` nicht ausführen müssen. Statt dessen können Sie direkt auf seine Mitglieder zugreifen.

Um die Rückgabe von Werten aus Funktionen zu erleichtern und Probleme beim Zugriff auf interne Strukturen des `zval`-Containers zu vermeiden, stehen eine Reihe von vordefinierten Makros zur Verfügung (wie gewöhlich). Diese Makros setzen automatisch den entsprechenden Typ und Wert, wie die Tabellen 9.14 und 9.15 zeigen.

Hinweis: Die Makros aus Tabelle 9.14 werden automatisch durch Ihre Funktion ausgegeben.

Makro	Beschreibung
`RETURN_RESOURCE(`*Ressource*`)`	Gibt eine Ressource zurück.
`RETURN_BOOL(`*bool*`)`	Gibt einen booleschen Wert zurück.
`RETURN_NULL()`	Gibt Nichts (einen `NULL`-Wert) zurück.
`RETURN_LONG(`*Long*`)`	Gibt einen Long zurück.
`RETURN_DOUBLE(`*Double*`)`	Gibt einen Double zurück.
`RETURN_STRING(`*String*`, duplicate)`	Gibt einen String aus. Der Flag `duplicate` gibt an, ob der String mit `estrdup()` dupliziert werden soll
`RETURN_STRINGL(`*String*`, `*Länge*`, duplicate)`	Gibt einen String der angegebenen Länge *Länge* aus; ansonsten verhält sich das Makro wie `RETURN_STRING`. Es ist jedoch schneller und binärsicher.

Tab. 9.14: Vordefinierte Makros zur Ausgabe von Werten aus einer Funktion

Makro	Beschreibung
RETURN_EMPTY_STRING()	Gibt einen leeren String aus
RETURN_FALSE	Gibt den booleschen Wert false aus
RETURN_TRUE	Gibt den booleschen Wert true aus

Tab. 9.14: Vordefinierte Makros zur Ausgabe von Werten aus einer Funktion

Hinweis: Die Makros in Tabelle 9.15 setzen nur den Ausgabewert, sie geben sie nicht aus Ihrer Funktion aus.

Makro	Beschreibung
RETVAL_RESOURCE(*Ressource*)	Setzt den Rückgabewert auf die angegebene Ressource.
RETVAL_BOOL(*bool*)	Setzt den Rückgabewert auf den angegebenen booleschen Wert.
RETVAL_NULL()	Setzt den Rückgabewert auf NULL.
RETVAL_LONG(*Long*)	Setzt den Rückgabewert auf den angegebenen Long.
RETVAL_DOUBLE(*Double*)	Setzt den Rückgabewert auf den angegebenen Double.
RETVAL_STRING(*String*, duplicate)	Setzt den Rückgabewert auf den angegebenen Sting und dupliziert ihn, wenn gewünscht, in den internen Zend-Speicher (siehe auch RETURN_STRING).
RETVAL_STRINGL(*String*, *Länge*,duplicate)	Setzt den Rückgabewert auf den angegebenen String und erzwingt dessen Länge entsprechend der Angabe in *Länge* (siehe auch RETVAL_STRING). Diese Makro ist schneller und binärsicher. Es sollte verwendet werden, wenn die Stringlänge bekannt ist.
RETVAL_EMPTY_STRING()	Setzt den Rückgabewert auf einen leeren String.
RETVAL_FALSE	Setzt den Rückgabewert auf den booleschen Wert false.
RETVAL_TRUE	Setzt den Rückgabewert auf den booleschen Wert true.

Tab. 9.15: Vordefinierte Makros zum Setzen des Rückgabewertes einer Funktion

Komplexe Typen, wie Arrays und Objekte, können über array_init() bzw. object_init() zurückgegeben werden, oder auch über die entsprechenden

Hash-Funktionen zu `return_value`. Da diese Typen nicht von einfachen Informationen konstruiert werden können, gibt es für sie keine vordefinierten Makros.

9.18 Ausdrucken von Informationen

Häufig ist es nötig, Meldungen im Ausgabestrom Ihres Moduls auszugeben, genau wie bei der Verwendung von `print()` in einem Skript. PHP bietet Funktionen für ganz allgemeine Aufgaben, wie etwa das Ausdrucken von Warnungen, Erzeugen von Ausgaben für `phpinfo()` usw. Die folgenden Abschnitte geben weitere Einzelheiten. Beispiele dieser Funktionen finden Sie auf der CD-ROM.

9.18.1 zend_printf()

`zend_printf()` funktioniert wie der Standard-`printf()`, außer dass es in den Ausgabestrom von Zend schreibt.

9.18.2 zend_error()

`zend_error()` kann zur Generierung von Fehlermeldungen verwendet werden. Diese Funktion hat zwei Argumente. Das erste gibt den Fehlertyp an (siehe `zend_errors.h`). Das zweite ist die eigentliche Fehlermeldung:

```
zend_error(E_WARNING, "This function has been called with empty
arguments");
```

Tabelle 9.16 zeigt eine Liste der möglichen Werte (siehe Abbildung 9.8). Auf diese Werte wird auch in `php.ini` Bezug genommen. Je nachdem welchen Fehlertyp Sie wählen, werden Ihre Meldungen protokolliert.

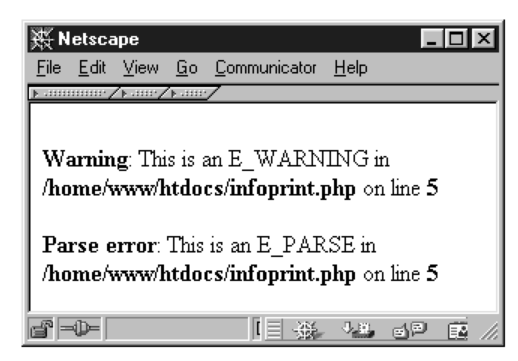

Abbildung 9.8: Anzeige von Warnungen im Browser

Fehler	Beschreibung
E_ERROR	Signalisiert einen Fehler und beendet sofort die Ausführung des Skripts.
E_WARNING	Signalisiert eine allgemeine Warnung, die Ausführung wird fortgesetzt.
E_PARSE	Signalisiert einer Parser-Fehler, die Ausführung wird fortgesetzt.
E_NOTICE	Signalisiert einen Hinweis, die Ausführung wird fortgesetzt. Beachten Sie, dass die Anzeige dieses Typs von Fehlermeldung in php.ini standardmäßig ausgeschaltet ist.
E_CORE_ERROR	Interner Fehler des Kerns, sollte nicht in vom Benutzer geschriebenen Modulen verwendet werden.
E_COMPILE_ERROR	Interner Fehler des Compilers, sollte nicht in vom Benutzer geschriebenen Modulen verwendet. werden
E_COMPILE_WARNING	Interne Warnung des Compilers, sollte nicht in vom Benutzer geschriebenen Modulen verwendet werden.

Tab. 9.16: vordefinierte Fehlermeldungen von Zend

9.18.3 Ausgabe einbinden in phpinfo()

Nachdem Sie ein reales Modul erstellt haben, möchten Sie möglicherweise Informationen über das Modul in `phpinfo()` ausgeben (zusätzlich zu dem Modulnamen, der standardmäßig in der Modulliste erscheint). Mit der Funktion `ZEND_MINFO()` können Sie in der `phpinfo()`-Ausgabe einen eigenen Abschnitt erzeugen. Diese Funktion sollte in den (bereits besprochenen) Deskriptorblock des Moduls eingefügt werden. Sie wird aufgerufen, wenn ein Skript `phpinfo()` aufruft.

Wenn Sie die Funktion `ZEND_MINFO` angeben, gibt PHP automatisch einen Abschnitt in `phpinfo()` aus, einschließlich des Modulnamens in der Überschrift. Alles andere müssen Sie selbst formatieren und ausdrucken.

In der Regel können Sie mithilfe von `php_info_print_table_start()` einen HTML-Tabellenkopf ausdrucken, und anschließend die Standardfunktonen `php_info_print_table_header()` und `php_info_print_table_row()` verwenden. Als Argumente geben Sie die Anzahl der Spalten (als Integer) und den Spalteninhalt (als String) an. Listing 9.14 zeigt eine Beispiel für einen Quellcode. Abbildung 9.9 zeigt die Ausgabe. Um den Tabellenfuß auszugeben, verwenden Sie `php_info_print_table_end()`.

```
php_info_print_table_start();
php_info_print_table_header(2, "First column", "Second column");
php_info_print_table_row(2, "Entry in first row", "Another entry");
```

```
php_info_print_table_row(2, "Just to fill", "another row here");
php_info_print_table_end();
```

Listing 9.14: Quellcode und Ausgabe in phpinfo()

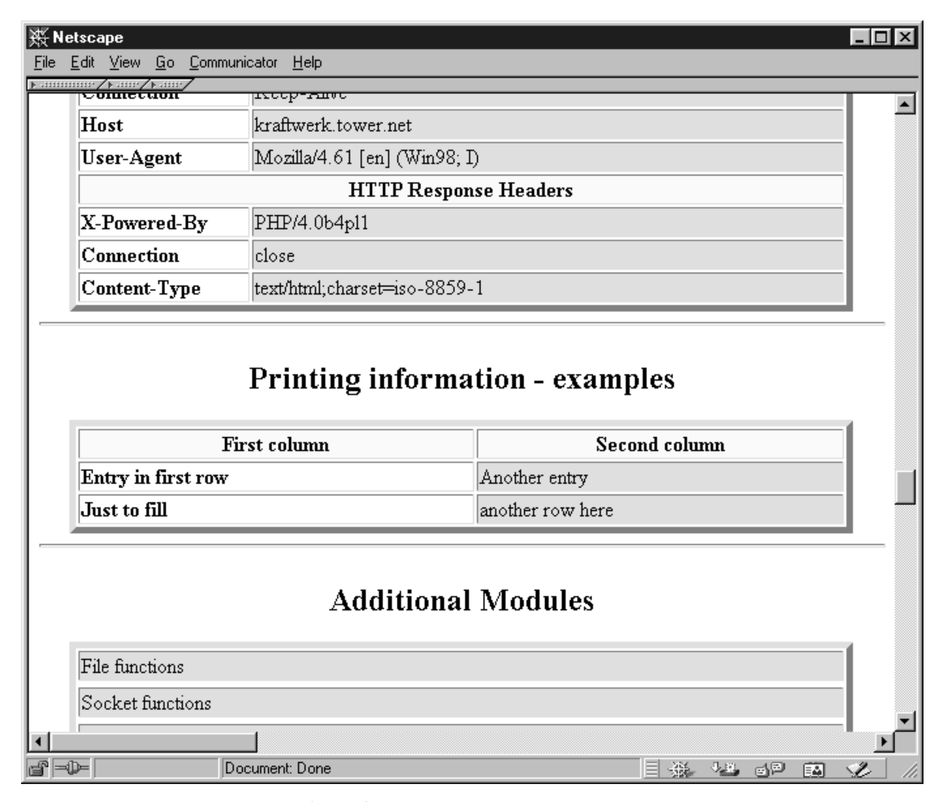

Abbildung 9.9: Ausgabe aus `phpinfo()`

9.18.4 Informationen über die Ausführung

Sie können auch Informationen über die Ausführung ausgeben, wie etwa über die Ausführung der aktuellen Datei. Der Name der Funktion, die gerade ausgeführt wird, kann über die Funktion `get_active_function_name()` ausgelesen werden. Diese Funktion gibt einen Zeiger auf den Funktionsnamen aus und hat keine Argumente. Um den Namen der gerade ausgeführten Datei auszulesen, verwenden Sie `zend_get_executed_filename()`. Diese Funktion greift auf die globalen Variablen des Executors zu, die über das Makro `ELS_C` an diesen übergeben wurden. Die globalen Variablen sind automatisch für alle Funktionen verfügbar, die direkt von Zend aufgerufen werden (sie sind Teil von `INTERNAL_FUNCTION_PARAMETERS`, das wir bereits früher in diesem Kapitel besprochen haben). Wenn Sie auf die globalen Variablen des Executors in

einer anderen Funktion zugreifen wollen, für die sie nicht automatisch verfügbar sind, rufen Sie in dieser Funktion einmalig das Makro `ELS_FETCH()` auf. Damit werden die Variablen einmal im lokalen Gültigkeitsbereich verfügbar.

Schließlich kann die derzeit ausgeführte Zeilennummer über die Funktion `zend_get_executed_lineno()` ausgelesen werden. Diese Funktion benötigt ebenfalls die globalen Variablen des Executors. Beispiele dieser Funktionen sehen Sie in Listing 9.15 und Abbildung 9.10. Natürlich sind alle Beispiele auch auf der CD-ROM verfügbar.

```
zend_printf("The name of the current function is %s<br>",
➥get_active_function_name());
zend_printf("The file currently executed is %s<br>",
➥zend_get_executed_filename(ELS_C));
zend_printf("The current line being executed is %i<br>",
➥zend_get_executed_lineno(ELS_C));
```

Listing 9.15: Ausgabe von Ausführungsinformationen

Abbildung 9.10: Ausgabe von Ausführungsinformationen

9.19 Initialisierungs- und Deinitialisierungsfunktionen

Initialisierungs- und Deinitialisierungsfunktionen können für einmalige Initialisierungen und Deinitialisierungen Ihre Module verwendet werden. Wie bereits in diesem Kapitel erwähnt (siehe die Beschreibung vom Deskriptorblock des Zendmoduls), gibt es globale, Modul- sowie Anfrageinitialisierungs- und Deinitialisierungsereignisse.

Die globale Initialisierungsfunktionen werden beim Hochfahren von PHP aufgerufen. Auf ähnliche Weise werden die globalen Deinitialisierungsfunktionen aufgerufen, sobald PHP beendet wird. Beachten Sie, dass sie tatsächlich nur einmal aufgerufen werden und nicht, wenn ein neuer Apache-Prozess generiert wird!

Die Initialisierungs- und Deinitialisierungsfunktionen des Moduls werden aufgerufen, wenn ein Modul geladen wird und initialisiert werden muss. Anfrageinitialisierungs- und Deinitialisierungsfunktionen werden aufgerufen, wenn eine Anfrage verarbeitet wird, also eine Datei ausgeführt wird.

Bei dynamischen Erweiterungen werden Module und Abfragen gleichzeitig initialisiert und deinitialisiert.

Die Deklaration und Einbindung dieser Funktionen geschieht über Makros. Einzelheiten hierüber finden Sie im Abschnitt »Deklarationen des Zend-Modulblocks«.

9.20 Aufruf von Benutzerfunktionen

Sie können Benutzerfunktionen aus Ihren eigenen Modulen heraus aufrufen. Dies ist sehr praktisch, wenn Sie Rückrufe realisieren wollen, z.B. zum Durchlaufen eines Arrays, zur Textsuche oder einfach für ereignisbasierte Programme.

Benutzerfunktionen können mit der Funktion `call_user_function_ex()` aufgerufen werden. Sie benötigt einen Hash-Wert für die Funktionstabelle, auf die Sie zugreifen möchten, einen Zeiger auf ein Objekt (wenn Sie eine Methode aufrufen möchten), den Funktionsnamen, den Rückgabewert, die Anzahl von Argumenten, das Argumentenarray und einen Flag, der angibt, ob Sie eine `zval`-Separation durchführen möchten:

```
ZEND_API call_user_function_ex(HashTable *function_table, zval *object,
                              ➥zval *function_name, zval **retval_ptr_ptr,
                              ➥int param_count, zval **params[]
                              ➥int no_separation);
```

Beachten Sie, dass Sie nicht beides, `function_table` und `object` angeben müssen; eines von beiden ist ausreichend. Wenn Sie eine Methode aufrufen möchten, müssen Sie das Objekt bereitstellen, welches die Methode enthält. In diesem Fall setzt `call_user_function()` die Funktionstabelle automatisch auf die Funktionstabelle des Objekts. Ansonsten brauchen Sie nur `function_table` anzugeben und können `object` auf `NULL` setzen.

Normalerweise ist die Standardfunktionstabelle die »Stamm«funktionstabelle, die alle Funktionseinträge enthält. Diese Funktionstabelle ist Bestandteil der globalen Variablen des Compilers. Der Zugriff erfolgt über das Makro `CG`. Um die globalen Variablen des Compilers Ihrer Funktion bekannzugeben, rufen Sie das Makro `CLS_FETCH` einmalig auf.

Der Funktionsname ist in einem `zval`-Container angegeben. Dies mag zunächst überraschend erscheinen, ist aber ein logische Schritt, da Funktionsnamen meistens als Parameter durch aufrufende Funktionen in Ihrem Skript ausgelesen werden, die wiederum in `zval`-Containern enthalten sind. Auf diese Weise müssen Sie Ihre Argumente nur durch diese Funktion übergeben. Dieses `zval` muss vom Typ `IS_STRING` sein.

Das nächst Argument besteht aus einem Zeiger auf den Rückgabewert. Sie brauchen keinen Speicher für diesen Container zuweisen, dies macht die Funktion selbst. Allerdings müssen Sie diesen Container (mithilfe von `zval_dtor()`) hinterher löschen!

Als nächstes folgen der Parameterzähler in Form eines Integers und ein Array, das alle notwendigen Parameter enthält. Das letzte Argument gibt an, ob die Funktion eine `zval`-Separation durchführen soll. Es sollte stets auf `0` gesetzt sein. Wenn es auf `1` gesetzt ist, belegt die Funktion zwar weniger Speicher, schlägt aber fehlt, sobald einer der Parameter getrennt werden muss.

Listing 9.16 und Abbildung 9.11 demonstrieren den Aufruf einer Benutzerfunktion. Der Programmcode ruft eine Funktion auf, die als Argument bereitgestellt wird, und übergibt den Rückgabewert dieser Funktion als eigenen Rückgabewert. Beachten Sie die Verwendung der Konstruktor- und Destruktoraufrufe am Ende. Möglicherweise ist es nicht notwendig, es auf diese Weise zu tun (da es sich hier um separate Werte handeln sollte, ist die Zuweisung wahrscheinlich sicher). Aber diese Vorgehensweise ist absturzsicher.

```
zval **function_name;
zval *retval;

if((ZEND_NUM_ARGS() != 1) || (zend_get_parameters_ex(1, &function_name)
➥!= SUCCESS))
{
    WRONG_PARAM_COUNT;
}

if((*function_name)->type != IS_STRING)
{
    zend_error(E_ERROR, "Function requires string argument");
}

CLS_FETCH();
```

```
if(call_user_function_ex(CG(function_table), NULL, *function_name, &retval,
➥0, NULL, 0) != SUCCESS)
{
    zend_error(E_ERROR, "Function call failed");
}

zend_printf("We have %i as type<br>", retval->type);

*return_value = *retval;
zval_copy_ctor(return_value);
zval_dtor(retval);

<?php

dl("call_userland.so");

function test_function()
{

    print("We are in the test function!<br>");

    return("hello");

}

$return_value = call_userland("test_function");

print("Return value: \"$return_value\"<br>");
?>
```

Listing 9.16: Aufruf von Benutzerfunktionen

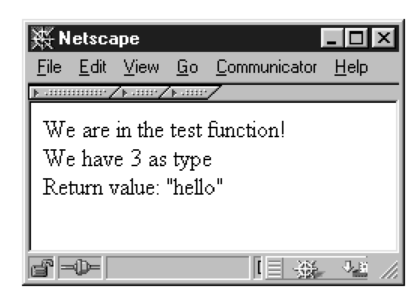

Abbildung 9.11: Aufruf von Benutzerfunktionen

9.20.1 Unterstützung für Initialisierungsdateien

PHP 4.0 bietet eine neu konzipierte Unterstützung für Initialisierungsdateien. Damit können Sie Standardinitialisierungseinträge direkt in Ihrem Code angeben, diese Werte zur Laufzeit lesen und ändern und Nachrichtenroutinen für Änderungsbenachrichtigungen schreiben.

Um einen .ini-Abschnitt in Ihrem eigenen Modul zu erstellen, verwenden Sie die Makros PHP_INI_BEGIN(), um den Beginn eines solchen Abschnitts zu markieren, und PHP_INI_END(), um das Ende zu markieren. Dazwischen können Sie PHP_INI_ENTRY() verwenden, um Einträge zu erzeugen.

```
PHP_INI_BEGIN()
    PHP_INI_ENTRY("first_ini_entry",  "has_string_value", PHP_INI_ALL,
                                                            ➥NULL)
    PHP_INI_ENTRY("second_ini_entry", "2",                PHP_INI_SYSTEM,
                                                            ➥OnChangeSecond)
    PHP_INI_ENTRY("third_ini_entry",  "xyz",              PHP_INI_USER,
                                                            ➥NULL)
                                                          PHP_INI_END()
```

Das Makro PHP_INI_ENTRY() hat vier Parameter: den Namen des Eintrags, den Wert des Eintrags, die Änderungsberechtigungen und einen Zeiger auf eine Änderungsbenachrichtigungsroutine. Sowohl der Name als auch der Wert des Eintrags müssen als String angegeben werden, ungeachtet dessen, ob sie tatsächlich Strings oder Interger sind.

Die Berechtigungen sind in drei Gruppen unterteilt: PHP_INI_SYSTEM ermöglicht eine Änderung nur direkt in der Datei php3.ini. PHP_INI_USER lässt während der Laufzeit das Überschreiben einer Änderung durch den Benutzer zu, und zwar über zusätzliche Konfigurationsdateien (wie etwa .htaccess). PHP_INI_ALL ermöglicht unbeschränkt Änderungen. Daneben gibt es noch eine vierte Ebene, PHP_INI_PERDIR. Deren Verhalten konnten wir jedoch bisher noch nicht herausfinden.

Der vierte Parameter besteht aus einem Zeiger auf eine Änderungsbenachrichtigungsroutine. Wenn einer dieser Initialisierungseinträge geändert wird, dann wird diese Routine aufgerufen. Solch eine Routine kann über das Makro PHP_INI_MH deklariert werden:

```
PHP_INI_MH(OnChangeSecond);             // handler for ini-entry
                                           ➥"second_ini_entry"

// specify ini-entries here

PHP_INI_MH(OnChangeSecond)
{
```

```
    zend_printf("Message caught, our ini entry has been changed to %s<br>",
            ➥new_value);

    return(SUCCESS);

}
```

Der neue Wert wird der Änderungsroutine als String in der Variable `new_value` übergeben. Wenn Sie sich die Definitionen von `PHP_INI_MH` ansehen, stellen Sie fest, dass Sie einige Parameter verwenden können:

```
#define PHP_INI_MH(name) int name(php_ini_entry *entry, char *new_value,
                                  ➥uint *new_value_length, void *mh_arg1,
                                  ➥void *mh_arg2, void *mh_arg3)
```

Diese Definitionen finden Sie alle in `php_ini.h`. Ihre Nachrichtenroutine hat Zugriff auf eine Struktur, die den gesamten Eintrag enthält, den neuen Wert, seine Länge und drei optionale Argumente. Diese optionalen Argumente können mit den zusätzlichen Makros `PHP_INI_ENTRY1` (das ein weiteres Argument zulässt), `PHP_INI_ENTRY2` (das zwei weitere Argumente zulässt) und `PHP_INI_ENTRY3` (das drei weitere Argumente zulässt) angegeben werden.

Die Änderungsbenachrichtigungsroutinen sollten eingesetzt werden, um Initialisierungseinträge lokal im Cache zu puffern, um einen schnelleren Zugriff zu erhalten oder bestimmte Aktionen durchzuführen, die erforderlich sind, wenn sich ein Wert ändert. Wenn beispielsweise ein Modul eine konstante Verbindung zu einem bestimmten Host benötigt und jemand den Namen des Host ändert, wird automatisch die alte Verbindung geschlossen und versucht, eine neue aufzubauen.

Der Zugriff auf die Initialisierungseinträge kann auch über die Makros erfolgen, die in Tabelle 9.17 aufgeführt sind.

Makro	Beschreibung
`INI_INT(`*Name*`)`	Gibt den aktuellen Wert des Eintrags `name` als Integer (Long) zurück.
`INI_FLT(`*Name*`)`	Gibt den aktuellen Wert des Eintrags `name` als Fließkommazahl (Double) zurück.
`INI_STR(`*Name*`)`	Gibt den aktuellen Wert des Eintrags `name` als String zurück. Hinweis: Dieser String wird nicht dupliziert, sondern zeigt statt dessen auf interne Daten. Bei weiteren Zugriffen ist eine Duplizierung in den lokalen Speicher erforderlich.

Tab. 9.17: Makros für den Zugriff auf Initialisierungseinträge in PHP

Makro	Beschreibung
INI_BOOL(*Name*)	Gibt den aktuellen Wert des Eintrags name als booleschen Wert zurück (definiert als zend_bool, das derzeit Zeichen ohne Vorzeichen bedeutet).
INI_ORIG_INT(*Name*)	Gibt den ursprünglichen Wert des Eintrags name als Integer (Long) zurück.
INI_ORIG_FLT(*Name*)	Gibt den ursprünglichen Wert des Eintrags name als Fließkommazahl (Double) zurück.
INI_ORIG_STR(*Name*)	Gibt den ursprünglichen Wert des Eintrags name als String zurück. Hinweis: Dieser String wird nicht dupliziert, sondern zeigt statt dessen auf interne Daten. Bei weiteren Zugriffen ist eine Duplizierung in den lokalen Speicher erforderlich.
INI_ORIG_BOOL(*Name*)	Gibt den ursprünglichen Wert des Eintrags name als booleschen Wert zurück (definiert als zend_bool, das derzeit Zeichen ohne Vorzeichen bedeutet).

Tab. 9.17: Makros für den Zugriff auf Initialisierungseinträge in PHP

Schließlich müssen Sie Ihre Initialisierungseinträge in PHP einbinden. Dies kann in den Initialisierungs- und Deinitialisierungsfunktionen des Moduls geschehen, in denen Sie die REGISTER_INI_ENTRIES() und UNREGISTER_INI_ENTRIES() verwenden:

```
ZEND_MINIT_FUNCTION(mymodule)
{

    REGISTER_INI_ENTRIES();

}

ZEND_MSHUTDOWN_FUNCTION(mymodule)
{

    UNREGISTER_INI_ENTRIES();

}
```

9.21 Wie geht es weiter?

Sie haben viel über PHP gelernt. Sie wissen nun, wie Sie dynamisch ladbare Module und statisch verknüpfte Erweiterungen erstellen. Sie haben gesehen, wie PHP und Zend mit der internen Speicherung von Variablen umgehen, und wie Sie diese Variablen generieren und auf sie zugreifen können. Sie ken-

nen eine Reihe von Toolfunktionen, die Ihnen viele Routineaufgaben, wie etwa die Ausgabe von informativen Texten, die automatische Einführung von Variablen in die Symboltabelle usw. abnehmen.

Auch wenn dieses Kapitel häufig einen »referenziellen« Charakter annahm, hoffen wir, dass Sie einen ersten Eindruck bekommen haben, wie Sie Ihre eigenen Erweiterungen schreiben können. Aus Platzgründen haben wir vieles weggelassen. Wir empfehlen Ihnen, sich die Zeit zu nehmen, die Headerdateien und einige Module näher anzusehen, insbesondere jene im Verzeichnis `ext/standard` und das MySQL-Modul, da diese allgemein bekannte Funktionen bieten. Dadurch erhalten Sie eine Vorstellung davon, wie andere Leute die API-Funktionen benutzt haben, speziell jene, die keinen Eingang in dieses Kapitel fanden.

9.22 Referenz: Einige Konfigurationsmakros

9.22.1 config.m4

Die Datei `config.m4` wird von `buildconf` verarbeitet und muss alle Anweisungen enthalten, die während der Konfiguration ausgeführt werden sollen. Hierzu können Befehle zur Überprüfung von externen Dateien, wie etwa Headerdateien, Bibliotheken usw. gehören. PHP definiert eine Reihe von Makros, die in diesem Prozess verwendet werden können. Die nützlichsten hiervon sind in Tabelle 9.18 beschrieben.

Makro	Beschreibung
`AC_MSG_CHECKING(`*Nachricht*`)`	Gibt während der Ausführung von `configure` einen überprüfenden <*Nachricht*>-Text aus.
`AC_MSG_RESULT(`*Wert*`)`	Gibt das Ergebnis von `AC_MSG_CHECKING` aus; als Wert sollte entweder `yes` oder `no` angegeben sein.
`AC_MSG_ERROR(`Nachricht`)`	Gibt die *Nachricht* während der Ausführung von `configure` als Fehlermeldung aus und bricht das Skript ab.
`AC_DEFINE(`*Name, Wert, Beschreibung*`)`	Fügt `#define` in `php_config.h` mit dem Wert `value` und einem Kommentar hinzu, der aus der *Beschreibung* besteht (dies ist sinnvoll bei bedingter Kompilierung Ihres Moduls).

Tab. 9.18: M4 Makros für config.m4

Makro	Beschreibung
AC_ADD_INCLUDE(*Pfad*)	Fügt einen Include-Pfad für einen Compiler hinzu; wird beispielsweise verwendet, wenn das Modul Suchpfade für Headerdateien hinzufügen muss.
AC_ADD_LIBRARY_WITH_PATH (*Bibliotheksname, Bibliothekspfad*)	Gibt eine zusätzliche, zu verknüpfende Bibliothek an.
AC_ARG_WITH(*Modulname, Beschreibung, nicht bedingter Test, bedingter Test*)	Ein recht mächtiges Makro, welches das Modul mit der *Beschreibung* in der Ausgabe configure --help hinzufügt. PHP überprüft, ob die Option with-<modulename> an das Configure-Skript übergeben wurde. Wenn ja, führt es das Skript den *nicht bedingten Test* aus (z. B.-with-myext=yes); in diesem Fall ist der Wert der Option in der Variablen $withval enthalten. Ansonsten führt es den *bedingten Test* aus.
PHP_EXTENSION(*Modulname*, [shared])	Diese Makro ist ein Muss für PHP, um Ihre Erweiterung zu konfigurieren. Sie können neben Ihrem Modulnamen ein zweites Argument liefern, das angibt, ob Sie ein Kompilierung als gemeinsam genutztes Modul (Shared Modul) beabsichtigen. Dies führt dazu, dass Ihre Quelle während der Kompilierung als COMPILE_DL_<*Modulname*> definiert wird.

Tab. 9.18: M4 Makros für config.m4

9.22.2 Zusätzliche API-Makros

Kurz vor der Freigabe dieses Buches wurden eine Reihe von Makros für die Zend-API eingeführt, die den Zugriff auf zval-Container vereinfachen (siehe Tabelle 9.19). Wir haben jedoch beschlossen, sie in den Beispielquellcodes nicht zu verwenden, da sie nicht häufig Zugriff auf zval-Container nehmen und die Makros zu Quellcode geführt hätten, der schwieriger zu lesen gewesen wäre.

Makro	Referenziert auf
Z_LVAL(zval)	(zval).value.lval
Z_DVAL(zval)	(zval).value.dval
Z_STRVAL(zval)	(zval).value.str.val
Z_STRLEN(zval)	(zval).value.str.len

Tab. 9.19: Neue API-Makros für den Zugriff auf zval-Container

Makro	Referenziert auf
Z_ARRVAL(zval)	(zval).value.ht
Z_LVAL_P(zval)	(*zval).value.lval
Z_DVAL_P(zval)	(*zval).value.dval
Z_STRVAL_P(zval_p)	(*zval).value.str.val
Z_STRLEN_P(zval_p)	(*zval).value.str.len
Z_ARRVAL_P(zval_p)	(*zval).value.ht
Z_LVAL_PP(zval_pp)	(**zval).value.lval
Z_DVAL_PP(zval_pp)	(**zval).value.dval
Z_STRVAL_PP(zval_pp)	(**zval).value.str.val
Z_STRLEN_PP(zval_pp)	(**zval).value.str.len
Z_ARRVAL_PP(zval_pp)	(**zval).value.hat

Tab. 9.19: Neue API-Makros für den Zugriff auf zval-Container

Updates zu diesem Kapitel finden Sie unter `http://www.phpwizard.net`.

Stichwortverzeichnis